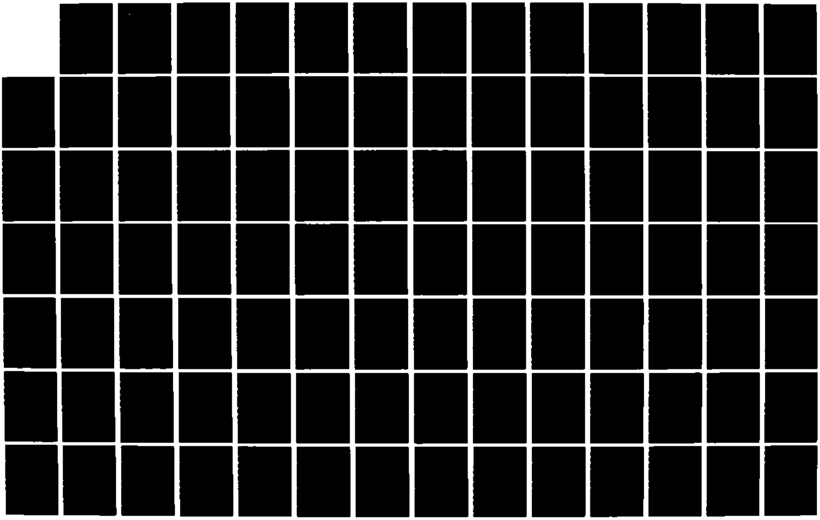
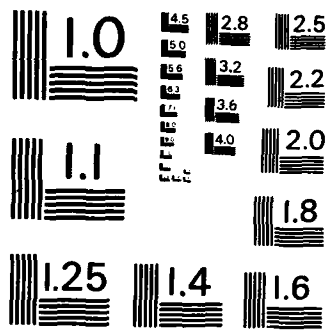
AD-A160 846 SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION STUDY(U) DELTA INFORMATION SYSTEMS INC HORSHAM PA AUG 85 NCS-TIB-85-6 DCA100-83-C-0047 1/2

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

2

NCS TIB 85-6

AD-A160 846

NATIONAL COMMUNICATIONS SYSTEM

TECHNICAL INFORMATION BULLETIN

85-6

SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION STUDY

AUGUST 1985

NCS TECHNICAL INFORMATION BULLETIN 85-6

SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION STUDY

AUGUST 1985

PROJECT OFFICER

EDWARD P. GREENE
Senior Electronics Engineer
Office of NCS Technology
and Standards

APPROVED FOR PUBLICATION:

Dennis Bodson
for MARSHALL L. CAIN
Assistant Manager
Office of NCS Technology
and Standards

FOREWORD

Among the responsibilities assigned to the Office of the Manager, National Communications System, is the management of the Federal Telecommunication Standards Program. Under this program, the NCS, with the assistance of the Federal Telecommunication Standards Committee identifies, develops, and coordinates proposed Federal Standards which either contribute to the interoperability of functionally similar Federal telecommunication systems or to the achievement of a compatible and efficient interface between computer and telecommunication systems. In developing and coordinating these standards, a considerable amount of effort is expended in initiating and pursuing joint standards development efforts with appropriate technical committees of the Electronic Industries Association, the American National Standards Institute, the International Organization for Standardization, and the International Telegraph and Telephone Consultative Committee of the International Telecommunication Union. This Technical Information Bulletin presents an overview of an effort which is contributing to the development of compatible Federal, national, and international standards in the area of facsimile standards. It has been prepared to inform interested Federal activities of the progress of these efforts. Any comments, inputs or statements of requirements which could assist in the advancement of this work are welcome and should be addressed to:

Office of the Manager
National Communications System
ATTN: NCS-TS
Washington, DC 20305
(202) 692-2124

SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION STUDY

August, 1985

Final Report

Submitted to:

NATIONAL COMMUNICATIONS SYSTEM

Office of Technology and Standards

Washington, DC 20305

Contracting Agency:

DEFENSE COMMUNICATIONS AGENCY

Contract Number - DCA100-83-C-0047

D E L T A   I N F O R M A T I O N   S Y S T E M S,   I N C.

Horsham Business Center, Bldg. 3

300 Welsh Road

Horsham PA 19044

Table of Contents

1.0 Introduction

This document summarizes work performed by Delta Information Systems, Inc. (DIS), for the National Communications System, an organization of the U. S. Government, under Task 2.0 of the Modification P00007 of Contract DCA100-83-C-0047. The purpose of this task was to investigate the efficiency and potential operational effectiveness of the Scan Line Difference Compression (SLDC) algorithm presented in Appendix A. Under this task, DIS performed a software simulation study of the SLDC algorithm. The software was run using selected CCITT binary standard images as input, and the results were compared with those of the MODREAD II compression algorithm run with the same input data. The MODREAD II algorithm was chosen for the comparison because it is the most effective compression technique currently available.

DIS has analyzed the SLDC algorithm and has generated tapes containing the Conditioned Image Files (CIF's) for the following twelve (12) combinations of the binary images and resolutions that were processed:

Image	Resolution
CCITT Image #1	200 lines/inch
CCITT Image #5	240 lines/inch
CCITT Image #7	300 lines/inch
	400 lines/inch

Each Subtask of Task 2.0 is presented in a section of this

report. Section 2.0 discusses the first subtask, in which Delta Information Systems studied a number of scan line conditioning techniques and generated the CIF's and corresponding prediction statistics using the technique selected in the study. In Section 3.0, the second subtask is presented, in which the FORTRAN software modules associated with the implementation the SLDC encoding algorithm are described both narratively and with structure charts, data flow charts, and software specifications.

Section 4.0 includes the third subtask, in which the software modules associated with the SLDC decoding algorithm are described; again, the software is described both narratively and with structure charts, data flow charts, and software specifications. The fourth subtask, in which Delta Information Systems ran the SLDC encoding program on each image with various sets of design parameters, is presented in Section 5.0. The compression statistics for all 168 simulation runs performed by Delta Information Systems are presented in this section.

2.0 Conditioned Image File Generation and Analysis

The objective of line conditioning is to increase the compressibility of the image under the constraint that the original image can be reconstructed from the CIF without distortion. Delta Information Systems analyzed a number of scan line conditioning techniques in this subtask and generated twelve Conditioned Image Files (CIF's), one for each image/resolution combination, with the technique selected. An example of a CIF is presented in Figure 2.1; the test image from which it was generated appears in Figure 2.2.

The conditioning technique selected by DIS predicts the binary state of an image element based on a weighted average of four of its close neighbors and produces a file of predictions, where correct predictons are '0's and incorrect predictions are '1's. Because each predicted element is based on the state of four neighboring elements (see Section A-2.1 in Appendix A), the predictor conditioning algorithm is completely described by a sixteen-entry state table; the state table employed in this study is presented in Table 2.1.

It was originally anticipated that each CCITT image would produce a different state table; however, this was not the case. DIS generated CIF prediction statistics for each file of the input image set. The results that were tabulated included the frequency of occurrence of each state together with the frequency of occurrence of a '0' pel for that state; this data revealed that all of the input images produced the same predictor conditioning algorithm, the one represented by the state table in Table 2.1.

Figure 2.1 - Conditioned Image File - Kanji 200lpi

CCITTの概要

沿革

CCITTは、国際電気通信連合（ITU）の四つの常設機関（事務総局、国際周波数登録委員会、CCIR、CCITT）の一つとして、ITUの中でも、世界の国際通信上の諸問題を真先に取上げ、その解決方法を見出して行く重要な機関である。日本名は、国際電信電話諮問委員会と称する。

CCITTの前身は、CCIF（国際電話諮問委員会）とCCIT（国際電信諮問委員会）である。CCIFは、1924年にヨーロッパに「国際長距離電話通信諮問委員会」が設置され、これが1925年のパリ電信電話会議のとき、正式に、「国際電話諮問委員会」として万国電信連合の公式機関となったものである。CCITは、同じく1925年の会議のとき、CCIFと併立するものとして設置された。

そして、CCIFは、1956年の12月に第18回総会が開催されたのち、CCITは、同年同月に第8回総会が開催されたのち、併合されて現在のCCITTとなった。このCCITTは、CCIFとCCITが解散した直後、第1回総会を開催し、第2回総会は、1960年にニューデリーで、第3回総会は、1964年、ジュネーブで、第4回総会は、1968年、アルゼンチンで開催された。

CCIFとCCITが合併したのは、有線電気通信の分野、とくに伝送路について電信回線と電話回線とを技術的に分ける意味がなくなってきたこと、各国とも大体において、電信部門と電話部門は同一組織内にあること、CCIFの事務局とCCITの事務局の合併による能率増進等がおもな理由であった。

CCITTは、上述のように、ヨーロッパ内の国ぐにによって、ヨーロッパ内の電信・電話の技術・運用・料金の基準を定め、あるいは統一をはかってきたので、現在でも、その影響を受け、会合参加国は、ヨーロッパの国が多く、ヨーロッパで生起する問題の研究が多い。たとえば、1960年のCCITT勧告の中で、技術上配慮する距離は約2,500kmであったが、これはヨーロッパ内領域を想定したものである。

しかしながら、1956年9月に敷設された大西洋横断電話ケーブルは、大陸間電話通信の自動化および半自動化への技術的可能性を与え、CCITTがこの問題を取り上げるに及び、CCITTの性格は漸次、汎世界的色彩を実質的に帯びるに至った。この汎世界的性格は第2次世界大戦後目ざましくなったアジア・アフリカ植民地の独立に伴ってITUの構成員の中にこれらの国が加わり、ITUの中に新しい意見が導入されたことにも起因して、技術面、政治面の双方から導入されてきた。CCITTの汎世界化は、1960年の第2回総会がニューデリーで開催されたことにもあらわれている。この総会までは、CCIT、CCIFのいずれにしろ、アメリカやアジアで総会が開催されたことがなく、CCITT委員長も、ニューデリー総会の準備文書で、この点には注目すべきであるとのべている。

任務

ITUは、全権委員会議、主管庁会議を始めとして、七つの機関をもち、それぞれの機関の権限と任務は国際電気通信条約に明記されている。そこで条約を参照してみるならば、CCITTの任務は、つぎのとおりとなっている。

「国際電信電話諮問委員会(CCITT)は、電信および電話に関する技術、運用および料金の問題について研究し、および意見を表明することを任務とする。」（1965年モントルー条約第187号）

「各国際諮問委員会は、その任務の遂行に当たって、新しい国または発展の途上にある国における地域的および国際的分野にわたる電気通信の創設、発達および改善に直接関連のある問題について研究し、および意見を作成するように妥当な注意を払わなければならない。」（同第188号）

「各国際諮問委員会は、また、関係国の要請に基づき、その国内電気通信の問題について研究し、かつ、勧告を行なうことができる。」（同第189号）

上記第187号と第188号にいわれる「意見」とは、フランス語の Avis から訳したもので、英語では、「勧告(Recommendation)」となっている。CCITTの表明する意見は、国際法的には強制力をもたないものであって、この点が、条約、電信規則、電話規則等各国を拘束する力をもっているものと異なる。もっとも意見とは称しても、技術的分野では、電信規則のごとき、各国政府が承認してその内容を実施する強制規則をもたないので、実際にある機器の仕様を定める場合には、多くの国の意見が統一されたこの「意見」に従わなければ、円滑な国際通信を行なうことができない場合が多い。この意見（または勧告）は、国際通信を行なう場合各国が直面する問題について、具体的意見を表明するもので、たとえば、大陸間ケーブルで大陸間通話を半自動化しようとする場合、その信号方式や取り扱う通話の種類および料金は、どのようにするかを研究して意見を表明する。したがって、CCITTの活動は、つねに時代の最先端を行くもので、CCITTの活動方向は、そのまま世界の国際通信の活動方向であるともいえる。

この意見は、また、電信規則以下のその他の規則のごとく、数年以上の間隔をもって開催される主管庁会議というような大会議の決定をまたなくても表明することができ、また、その改正も容易であるので、現在のように進歩の早い国際通信界では、関係国の意見を統一した国際的見解としては非常に便利である。

Figure 2.2 - Original Image File - Kanji 200lpi

Table 2.1 - Predictor Conditioning Algorithm State Table

A B C D	PREDICTION
0 0 0 0	0
0 0 0 1	1
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	1
1 0 0 0	0
1 0 0 1	0
1 0 1 0	0
1 0 1 1	1
1 1 0 0	0
1 1 0 1	1
1 1 1 0	0
1 1 1 1	1

DIS generated run length statistics using the selected predictor conditioning algorithm. For each CIF, a set of statistics was tabulated; the statistics generated include the following:

1. Number of bits in CIF;
2. Weight of CIF;
3. Histogram of the weight of the Conditioned Scan Lines (CSL's) within the CIF;
4. Histogram of run lengths in the CIF; and,
5. Two histograms of the weight of CIF segments, where each CIF segment represents a contiguous 32-bit or 64-bit section of the CSL.

Delta Information Systems generated a Conditioned Image File for each test image at each resolution in the input set. The set of twelve CIF's was written onto a 9-track, 1600 BPI magnetic tape as described in the SOW; DIS will deliver two copies of this tape. The operating instructions for the software modules employed to generate the CIF's and their associated statistics are included in Appendix B; the code listings for these modules appear in Appendix C.

3.0 SLDC Encoder Software Design

Scan Line Difference Compression is an encoding method employed in the transmission of binary images. The SLDC encoder program compresses the Conditioned Image Files (CIF's) generated in the previous subtask by taking advantage of the local conditional statistics of each image. The SLDC algorithm is described in detail in Appendix A; a brief description of the encoding process is presented here, along with the documentation for the software modules associated with the encoder program. The operating instructions for the encoder program are presented in Appendix B; the code listings appear in Appendix C.

3.1 Functional Description

File encoding is done one line at a time. As each Conditioned Scan Line (CSL) is read in, the Hamming weight of the line is calculated. If the line weight is zero, the line is encoded with a single bit, set to one (1). If the line weight is greater than zero, the integer value of the line weight is placed in the first thirteen bits of the Encoded Scan Line (ESL) (the integer value of the line weight does not exceed 2^{12}) and the CSL is encoded as a series of n-length segments, where n is the user-defined segment length. The codewords for all possible segment weights which can occur on a line of the current line weight are determined from the statistics of the line; a Huffman coding technique, illustrated in Figure 3.1, is employed to generate the codewords.

The next step is to partition the CSL into segments of the length

Huffman Probability Table Reduction Process

1	2	3	4	5	6
p1	p1	p1	p1	p1	>p1
p2	p2	>p2	>p2	>p2	p2
				+}	
p3	>p3	p3	p3	p3	
			+}		
p4	p4	p4	p4		
		+}			
p5	p5	p5			
	+}				
p6	p6				
+}					
p7					

Huffman Codeword Table Expansion Process

1	2	3	4	5	6
1	1	1	1	1	{ 0
001	001	{000	{01	{00 <	1
011	{010	001	000 <	01 <	
0000	011	010 <	001 <		
0001	0000 <	011 <			
0100 <	0001 <				
0101 <					

Figure 3.1 - Huffman Codeword Assignment Process

specified by the operator. Each segment is encoded with a two-part Segment State Word (SSW) that consists of a segment weight codeword and a segment rank. As each segment is extracted from the CSL, it is placed into a buffer and its weight is calculated. If the segment weight is zero, the zero-length codeword is placed into the SSW and the segment rank, which is an integer value which indicates the positions of the ones (1's) in the segment, is omitted.

If the segment weight is greater than zero and less than the maximum weight allowable for a segment of that size, the rank encoding procedure is initiated. The segment buffer is examined from left to right to determine the rank of the segment using the method described in Appendix A. Once the rank is determined, it is placed, along with the segment weight codeword, in the output buffer using the prescribed number of bits generated by the rank length equation.

If the segment weight is greater than the maximum allowable weight, the rank encoding procedure is not invoked; the SSW for the segment is comprised of the codeword for the "exceeded maximum" condition followed by the segment in uncompressed form. Maximum segment weight values must be employed in order to avoid arithmetic overflow in the rank encoding computations. The maximum segment weights for the various segment lengths are presented in Table 3.1; these values are slightly different from those found in Appendix A because the HP-1000 simulation system employs a sign-bit in its 32-bit double integer word. Therefore, the HP-1000 has a maximum integer value of 2^{31} instead of the value of 2^{32} apparently employed to construct Table A-1.

Segment Length in Bits	Maximum Segment Weight Value	
	16-bit Word Length	32-bit Word Length
≤18	unconstrained	unconstrained
20	6	"
24	5	"
28	4	"
32	4	"
36	4	12
40	3	10
44	3	9
48	3	9
52	3	8
56	3	8
60	3	7
64	3	7

Table 3.1 - Maximum Weight Segment for Normal SLDC Processing

For each segment encoded, the segment weight code length and the rank length are added to the total number of bits encoded to give the compression ratio of either the entire image or, if requested, of each line.

3.2 Software Documentation

The software documentation for the SLDC encoder program is presented in this section and includes a structure chart, Nassi-Schneiderman flow charts for the major software modules, and descriptions of the functions and data storage methods associated with the encoder program. Also included in this section is the data flow diagram for the overall simulation project and Nassi-Schneiderman flow charts for the software modules that generate the Conditioned Image Files; this additional documentation is not directly related to the SLDC encoder program, but is presented here in order to convey a clearer picture of the software simulation of the SLDC compression algorithm.

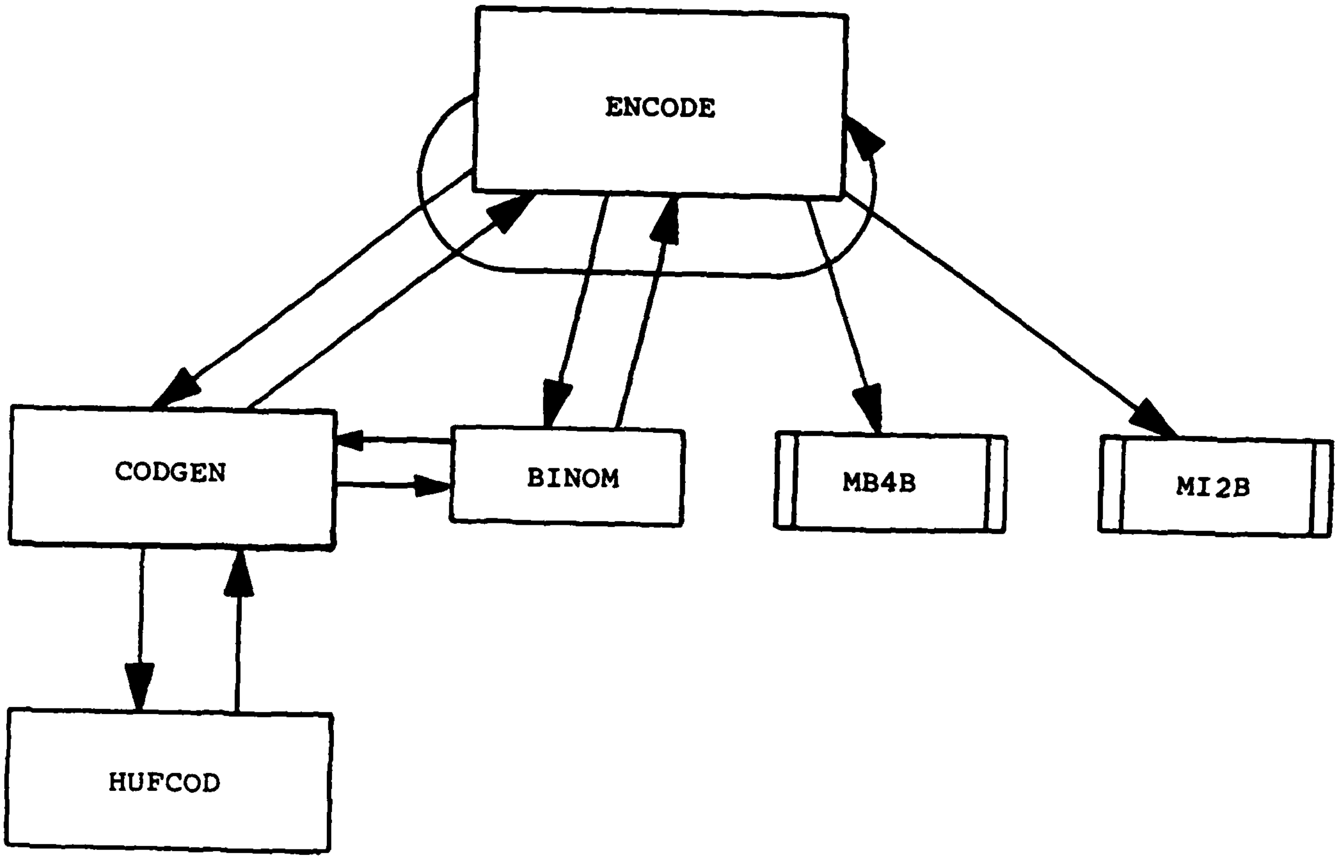


Figure 3.2 - Structure Chart for Module: ENCODE

Figure 3.3 - ENCODE.FTN - SLDC Encoding Program

OPEN files and read input parameters

DO for # records in file

READ Conditioned Scan Line (CSL) into buffer

Calculate line weight

Line weight = 0 ?

NO

Put line weight in first 13 bits of ESL

Encoded Scan Line (ESL)

CALL CODGEN with lineweight and max weight

of segment

DO for # segments in CSL or

"1"s seen = line weight

Put segment of segment length from

line buffer into segment buffer

Calculate segment weight

YES

Set first bit

of first word

of ESL and

increment

encoded line

weight count

Figure 3.3 - ENCODE.FTN - SLDC Encoding Program

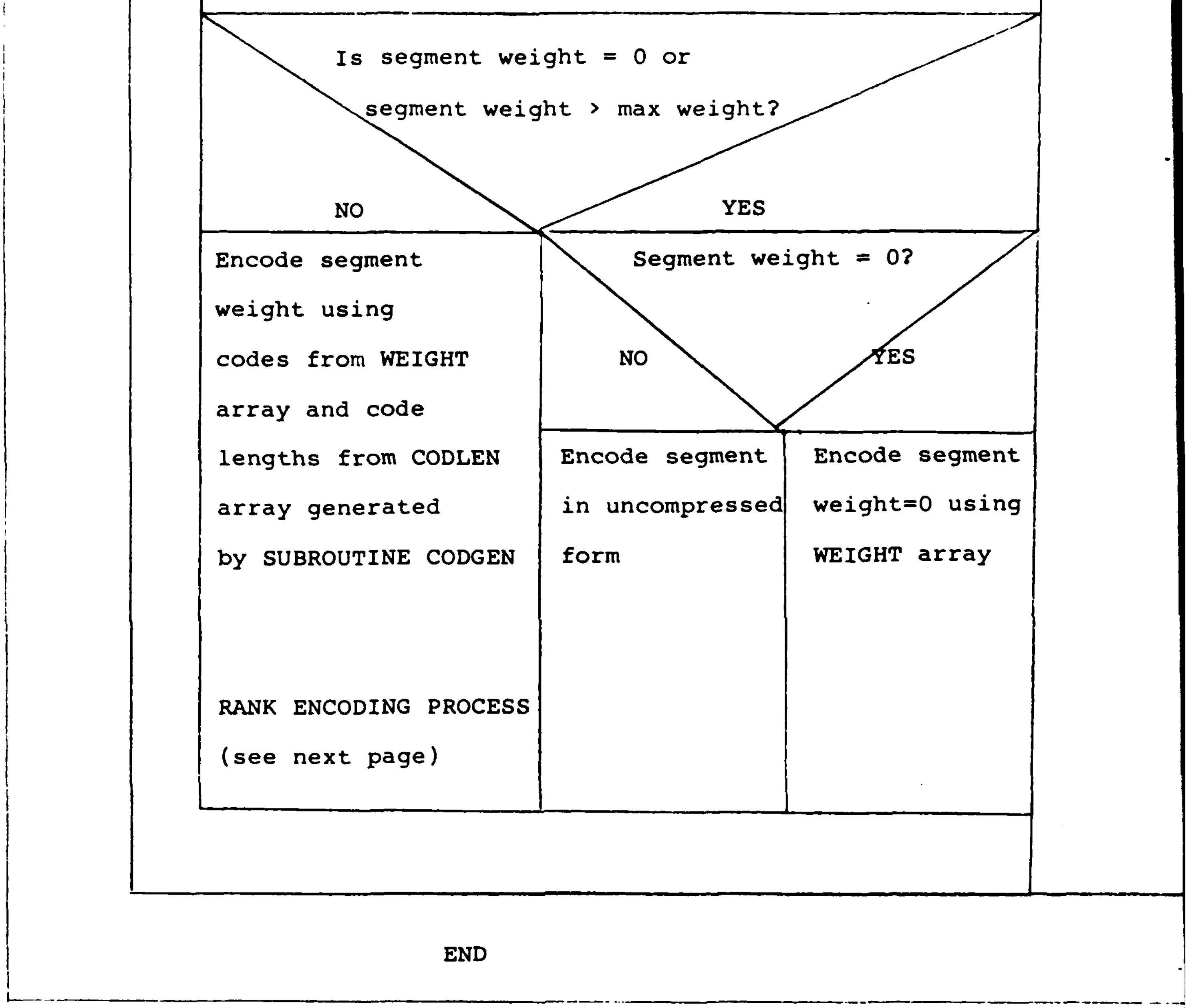


Figure 3.3 - ENCODE.FTN - SLDC Encoding Program

RANK Encoding Process

DO for # bits in segment OR "1"s seen = segment weight
OR rest of segment is filled with "1"s

- Put present buffer word into an integer
- DO for # bits in segment OR "1"s seen = segment weight
  OR rest of segment is filled with "1"s
  - Only "1"s remaining or "1"s seen = segment weight?
    - NO:
      - Present bit = "1"?
        - YES:
          - Add answer from RANK constant equation described in SOW to rank
          - Increment "1"s seen
    - YES:
      - Set flag to exit loop

Use RANK length equation described in SOW to determine RANK length

Encode RANK length number of bits to encode rank in ESL

Figure 3.4 - CODGEN.FTN - SLDC Huffman Coding Routine

DO for Weight = 0 to Max weight

> Calculate probabilities for each weight
> using EQ [A-4] in Appendix A

Assign Probability to Maxweight + 1
(1 - sum of probabilities)

Bubble sort probabilities in descending order keeping
track of original positions

CALL HUFCOD to calculate codewords and codelengths
for probabilities

DO for weight = 0 to Maxweight +1

> Put corresponding weight value back into proper weight
> subscript of Weight array and Code length array
> to provide easy access to codes and code lenghts

RETURN

Figure 3.5 - HUFCOD.FTN - Huffman Codword Generator

DO for 0 to Max weight of segment

- Add last two entries in column
- Determine it's rank in next column
- Place it in the next column at it's proper rank
- Mark the position in the position array

Initialize codes and indicies

DO for Max weight down to 0

- Add a bit to codeword at next column's position according to position array
- Set new bit of last entry in column to "0"
- Set new bit of last entry in new column to "1"

RETURN

Program Documentation for module: ENCODE

PROGRAM: ENCODE

DESCRIPTION: This program encodes an input Conditioned Image File using SLDC encoding techniques. The program interactively inquires for, then accepts input parameters used in runs having different file sizes and design parameter sets. A summary of each run is printed including: names and sizes of files used and compression statistics. Options to create an output file (and/or) print line by line compression statistics are also included.

RUNSTRING: ENCODE,<INPUT NAME>,<OUTPUT NAME>

INPUT NAME — Input Image File name

OUTPUT NAME — Output Encoded File name.(Optional)

ORDER OF INPUT PARAMETERS:

1) # Records to be output
2) Decision to create output file
3) Maximum weight for segment length
4) CCITT file number
5) File resolution
6) Arithmetic word length
7) # Words per input record
8) # Records per input file
9) Segment length for compression
10) Decision to print line by line compression statistics

MODULES CALLED:

CODGEN — Huffman code generation subroutine. Generates Huffman codes for each line in input file with line weight greater than zero.

BINOM — Binomial coefficient function

MI2B — Subroutine to place an integer into a given position in an array

MB4B — Subroutine to move parts of one array into parts of another array.

MVBITS BTEST,IBSET	FORTRAN bit manupulating routines

NAMED COMMON
DESCRIPTIONS:

Block Name: ENCOD
Module Common to: CODGEN

Descriptions:

WEIGHT	Array of Huffman codes returned from CODGEN with array subscripts denoting segment weights
SEGL	Segment length design parameter used in EQ. [A-4].
NUMBER	Number of bits per CSL to be used in EQ. [A-4].
CODLEN	Code length array returned from CODGEN with values corresponding to those of WEIGHT array

Subroutine Documentation for module: CODGEN

SUBROUTINE: CODGEN

MODULES CALLED FROM: ENCODE and DECODE

PURPOSE: This subroutine generates probabilities for each possible weight from 0 upto and including the maximum weight allowable for a given segment+1 passed by parameter. These probabilities are then used by SUBROUTINE HUFCOD to generate codes and code lenghts for these probabilities.

MODULES CALLED:

BINOM	Binomial coefficient function
HUFCOD	Subroutine which assigns variable length Huffman codes and code lengths to segment weights which are arranged in most probable to least probable order.

CALLING FORMAT: CALL CODGEN(LNWGT,MAXWGT)

ARGUMENT DESCRIPTIONS:

LNWGT	Current CSL weight generated by ENCODE. LNWGT is used by EQ. [A-4].
MAXWGT	Maximum segment weight encodable for design parameter segment length taken from TABLE [A-1] MAXWGT is also used by EQ. [A-4].

NAMED COMMON DESCRIPTIONS:

Block Name: ENCOD
Module Common to: ENCODE

Descriptions:

WEIGHT	Array of Huffman codes returning to ENCODE with array subscripts denoting segment weights

Subroutine Documentation for module: CODGEN

SEGL	Segment length design parameter used in EQ. [A-4].
NUMBPL	Number of bits per CSL to be used in EQ. [A-4].
CODLEN	Code length array returning to ENCODE with values corresponding to those of WEIGHT array

Block Name: HUFBLK
Module Common to: HUFCOD

Descriptions:

PROB	Probability array containing probabilities of segment weights of a given CSL.
CODE	Array returned holding huffman codes in most probable to least probable order
MAXWT	Maximum weight of a segment. (described above)
CDWLEN	Code length array matching values in CODE array.

Subroutine Documentation for module: HUFCOD.FTN

SUBROUTINE: HUFCOD

MODULE CALLED FROM: CODGEN

PURPOSE: This subroutine takes probabilities generated by CODGEN and determines the positions for a right to left expansion of the huffman coding table according to the values of the probabilities.

CALLING FORMAT: CALL HUFCOD

NAMED COMMON DESCRIPTIONS:

Block Name: HUFBLK
Module Common to: CODGEN

Descriptions:

PROB	Probability array containing probabilities of segment weights of a given CSL.
CODE	Array returning huffman codes in most probable to least probable order
WGTNUM	Maximum weight of a segment. (described above)
CDWLEN	Code length array matching values in CODE array.

Subroutine Documentation for module: BINOM.FTN

FUNCTION: BINOM

MODULES CALLED FROM: CODGEN,ENCODE,DECODE

PURPOSE: This function uses an algorithm found in CACM to calculate the binomial coefficient given two parameters. The maximum value returned is 2^{31} which is the maximum value representable in a FORTRAN DOUBLE INTEGER.

CALLING FORMAT: n = BINOM(N,M) where n is a single or double integer

ARGUMENT DESCRIPTIONS:

N,M N objects taken M at a time

DIS FORTRAN 77 bit handling routines

FUNCTION: Provides facilities for storing integers, in packed form, in variables and arrays, and for retrieving them at a later time.

CALLS: CALL MI2B(I2, BA, JB, NB)

Stores the INTEGER*4 I2 into BA, starting at the JBth bit, and occupying NB bits.

I4B(BA, JB, NB)

Retrieves (as its functional value) the INTEGER*4 integer stored in BA, starting at the JBth bit, and occupying NB bits. DI4B returns a INTEGER*4 integer value and must be declared as such in the calling routine.

CALL MB4B(TBA, JTB, NB, FBA, JFB)

Replaces JTBth through the (JTBth + NB - 1)st bits of TBA with the JFBth through the (JFB + NB - 1)st bits of FBA.

ARGUMENTS:

I2	- INTEGER*4 variable or array element
JTB,JFB,JB	- INTEGER*2 starting bit position for string; must be > 0.
NB	- INTEGER*2 no. of bits in a string (i.e. string size must be > 0.
TBA,FBA,BA	- INTEGER*2 or INTEGER*4 arrays used for storing "packed" bit strings.

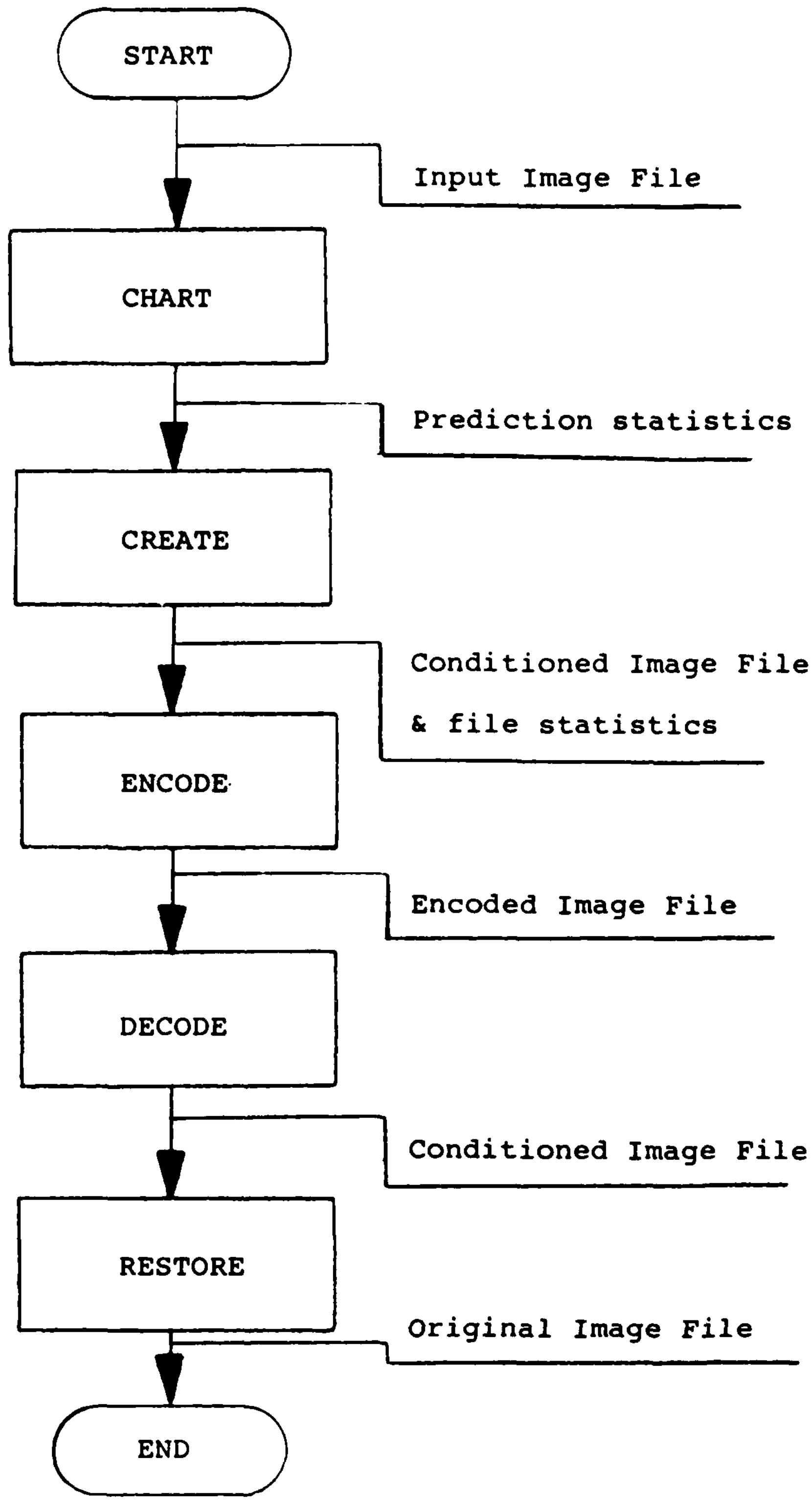


Figure 3.6 - SLDC Data Flow Diagram

Figure 3.7 - CHART.FTN - SLDC Probability Statistics Generator

OPEN files and read input parameters

Initialize all count arrays and buffers

DO for # records in file

- DO for # words in input record
  - DO for # bits per word
    - Test for value of current bit being examined
    - Test for neighboring bit values
    - Search template array for current neighbor template
    - Increment count array for total, and black white count of current neighbor template

Print all statistics

Figure 3.7 - CHART.FTN - SLDC Probability Statistics Generator

WRITE proper prediction (1 or 0) for each neighbor template

to probability file for use in later programs

E N D

Figure 3.8 - CREATE.FTN - SLDC CIF Generating Program

OPEN files and read input parameters

READ Probability Data Generated from PROGRAM CHART

Initialize buffers and count arrays

DO for # records in file

DO for # words in input record

DO for # bits per word

Test for neighboring bit values

Search probability data for proper neighbor template and respective prediction

Is prediction correct?

NO

Set current bit to "1"

Increment lineweight count, 32 bit segment weight count, and 64 bit segment weight count

Figure 3.8 - CREATE.FTN - SLDC CIF Generating Program

Reach end of 32 bit segment?

YES

Increment proper 32 bit weight subscript in 32 bit count array

Reach end of 64 bit segment?

YES

Increment proper 64 bit weight subscript in 64 bit count array

Put present output word in output buffer

Increment proper line weight subscript in line weight count array

Add line weight to file weight

WRITE output buffer to output file

Print out all statistics

CLOSE files

E N D

Figure 3.9 - RESTORE.FTN - SLDC Original Restoration Program

OPEN files and read input parameters

READ Probability Data Generated from PROGRAM CHART

Initialize buffers and count arrays

DO for # records in file

DO for # words in input record

DO for # bits per word

Test current bit being examined

Test for neighboring bit values

Search probability data for proper neighbor template and respective prediction

Is current bit "1"?

YES	NO
Place bit opposite of prediction bit in current bit position	Place prediction bit in current bit position

Figure 3.9 - RESTORE.FTN - SLDC Original Restoration Program

Put present output word in output buffer

WRITE output buffer to output file

CLOSE files

E N D

4.0 SLDC Decoder Software Design

The SLDC decoder program reconstructs the Conditioned Image Files (CIF's) compressed in the previous subtask by employing the identical codeword generator and rank encoding procedure employed in the SLDC encoder program. A brief description of the decoding process is presented here, along with the documentation for the software modules associated with the decoder program. The operating instructions for the decoder program are presented in Appendix B; the code listings appear in Appendix C.

4.1 Functional Description

Decoding is also done one line at a time. Each ESL is read into a buffer and the first bit in the line (bit 15 in word 1) is checked. If the bit is set to one (1), the line weight is zero and all bits in the decoded CSL are set to zero (0). If the bit is not set to one (1), the first 13 bits of the ESL are interpreted as the line weight and the Huffman coding subroutine is called to generate Huffman codes for all possible segment weights for that particular line weight. The program proceeds by decoding one segment at a time; a number of bits equal to the maximum segment weight codeword are read from the ESL into a buffer. The segment weight is then decoded by comparing each code in the Huffman code array with the same number of bits in the codeword buffer until the code is found.

If the segment weight is found to be zero, the next n bits in the decoded CSL, where n is the segment length, are set to zero (0). If the segment weight is found to be greater than the maximum allowable

weight, the segment has been "transmitted" uncompressed, and the next n bits in the ESL are placed directly into the decoded CSL, where, again, n is the segment length.

For the remaining condition, in which the segment weight is greater than zero and less than the maximum allowable, the length of the rank can be calculated by using the segment weight in the rank length equation. The rank is then determined from the integer value of the next r bits in the ESL, where r is the determined rank length. The segment is reconstructed by placing a one (1) in the output segment buffer whenever the difference between the rank and the calculated binomial coefficient (see Appendix A) is positive; when a one (1) is found, the rank is decremented by an amount equal to the coefficient. This rank decoding process continues until the entire rank is examined, or the rank is zero after the decrement takes place. The output segment buffer is then placed in the decoded CSL. The segment decoding continues until all segments in the line have been decoded, or until the sum of the segment weights of the segments placed in the decoded CSL is equal to the line weight.

4.2 Software Documentation

The software documentation for the SLDC decoder program is presented in this section and includes a structure chart, a Nassi-Schneiderman flow chart, and a description of the functions and data storage methods associated with the main software module, DECODE. All other modules associated with the decoder program are identical to those of the encoder program; the documentation for these modules appears in section 3.2.

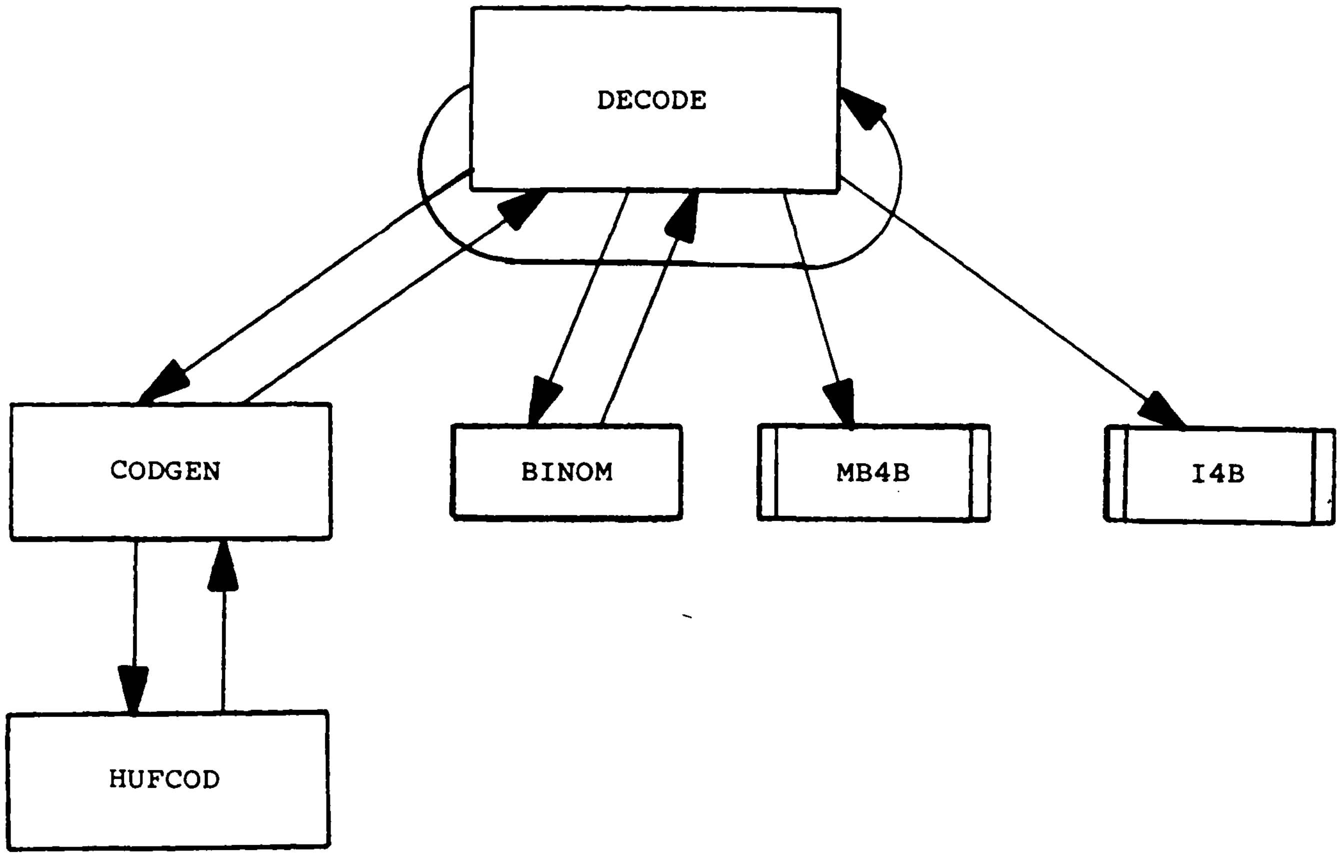


Figure 4.1 - Structure Chart for Module: DECODE

Figure 4.1 - DECODE.FTN - SLDC Decoding Program

OPEN files and read input parameters

DO for # records in file

READ ESL into buffer

First bit in first word of ESL set ?

NO

CALL CODGEN with lineweight and max weight of segment

DO WHILE # of "1"s put in output line < line weight

READ in code buffer (double integer from ESL)

DO WHILE segment weight code is not found

Check current code in array with same number of bits in code buffer

Figure 4.1 - DECODE.FTN - SLDC Decoding Program

Is segment weight = Sk'

NO

Plug line weight into RANK length equation to get rank length

Use that many bits of ESL and place decoded rank in output buffer

YES

Place uncoded segment in output buffer

END

Program Documentation for module: DECODE

PROGRAM: DECODE

DESCRIPTION: This program decodes an input Encodes Image File coded using SLDC encoding techniques. The program interactively inquires for, then accepts input parameters used in runs having different file sizes and design parameter sets. A summary of each run is printed including names and sizes of files used and design parameters.

RUNSTRING: DECODE,<INPUT NAME>,<OUTPUT NAME>

INPUT NAME	Input Image File name
OUTPUT NAME	Output Conditioned Image File

ORDER OF INPUT PARAMETERS:

1) # Words per output record
2) # Records to be output
3) Arithmetic word length
4) # Words per input record
5) # Records per input file
6) Segment length for compression
7) Maximum weight per segment

MODULES CALLED:

CODGEN	Huffman code generation subroutine. Generates Huffman codes for each line in input file with line weight greater than zero.
BINOM	Binomial coefficient function
I4B	Functionqto extract an integer from a given position in an array
MB4B	Subroutine to move parts of one array into parts of another array.
MVBITS,IBITS BTEST,IBSET	FORTRAN bit manupulating routines

NAMED COMMON DESCRIPTIONS:

Block Name: ENCOD

Program Documentation for module: DECODE

Module Common to: CODGEN

Descriptions:

WEIGHT	Array of Huffman codes returned from CODGEN with array subscripts denoting segment weights
SEGL	Segment length design parameter used in EQ. [A-4].
NUMBER	Number of bits per CSL to be used in EQ. [A-4].
CODLEN	Code length array returned from CODGEN with values corresponding to those of WEIGHT array

5.0 SLDC Compression Results and Analysis

Simulation runs made on each image using the SLDC technique included design parameters of both 16 and 32 bit arithmetic word lengths using segment lengths from 8 to 64, in increments of 8. The parameters were chosen to experiment with many combinations of word lengths and segment lengths to determine the most efficient parameter set for each image at each resolution.

The results from the SLDC compression simulation runs showed the segment lengths using the 32 bit arithmetic word to have consistently higher compression on each image tested at all four resolutions, except for segment lengths of 8 and 16, where compression is equal due to the design of the algorithm. The higher compression in the 32 bit word simulation runs is related to the maximum weight constraint difference between the 16 and 32 bit word lengths, where compression of the segment to be encoded is bypassed if the weight of the segment exceeds the maximum weight constraint. The compression bypass involves skipping the rank encoding part of the algorithm when the segment weight is higher than the maximum weight allowable for that segment. Another direct result of the compression bypass is a difference in processing time, which is consistently lower in the 16 bit word simulation runs due to the processing skipped when the rank encoding is omitted. The processing times of all of the simulation runs appear to be directly related to the maximum weight constraint and almost independent of segment length, which illustrates which parameter determines the difference in the complexity of the algorithm between simulation runs.

The comparison of the overall compression between images was as expected, with the English letter compressing the best, followed by the French journal, and then the Kanji text; the compression statistics are presented in Tables 5.1 through 5.12, and are illustrated graphically in Figures 5.1 through 5.12 (In all of the compression graphs, the □ indicates results of the 16 bit word length runs and the + indicates the results of the 32 bit word length runs.). The French and English document compression curves are quite similar, with maximum compression of both 16 and 32 bit word length runs occurring at the same segment lengths. It appears that the similarities between the two compression curves are a result of the similar probability distributions of the two files along with similar CIF constructions. The Kanji compression curve is different than those of the French and English documents. The Kanji 32 bit word curve is without the "peaking" effect demonstrated by the French and English curves. The maximum compression point is virtually indistinguishable in the 24 to 56 bit segment range, as compared to the definite peaks in both the English and French curves.

The maximum compression point varies with resolution. As resolution increases, the segment length of the maximum point of the 16 and 32 bit word length run curves increases. This can be described as a "right shift" of the peaks in the graphs. The shift is due to more segments with lower segment weights as the resolution increases; therefore, fewer segments exceed the maximum weights at higher segment lengths, and this leads to optimum compression at higher segment lengths. The suggested set of parameters to achieve maximum compression, taking processing time into consideration, is presented in Table 5.13.

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	15.31	10.31
16	16	18.49	11.22
16	24	19.41	9.41
16	32	17.94	9.23
16	40	15.67	9.04
16	48	14.77	8.56
16	56	14.14	8.50
16	64	13.33	8.23
32	24	19.53	14.04
32	32	20.03	17.21
32	40	20.17	11.34
32	48	19.89	10.22
32	56	19.11	10.03
32	64	17.61	9.39

Table 5.1 - Compression Stats, English Letter - 200 LPI

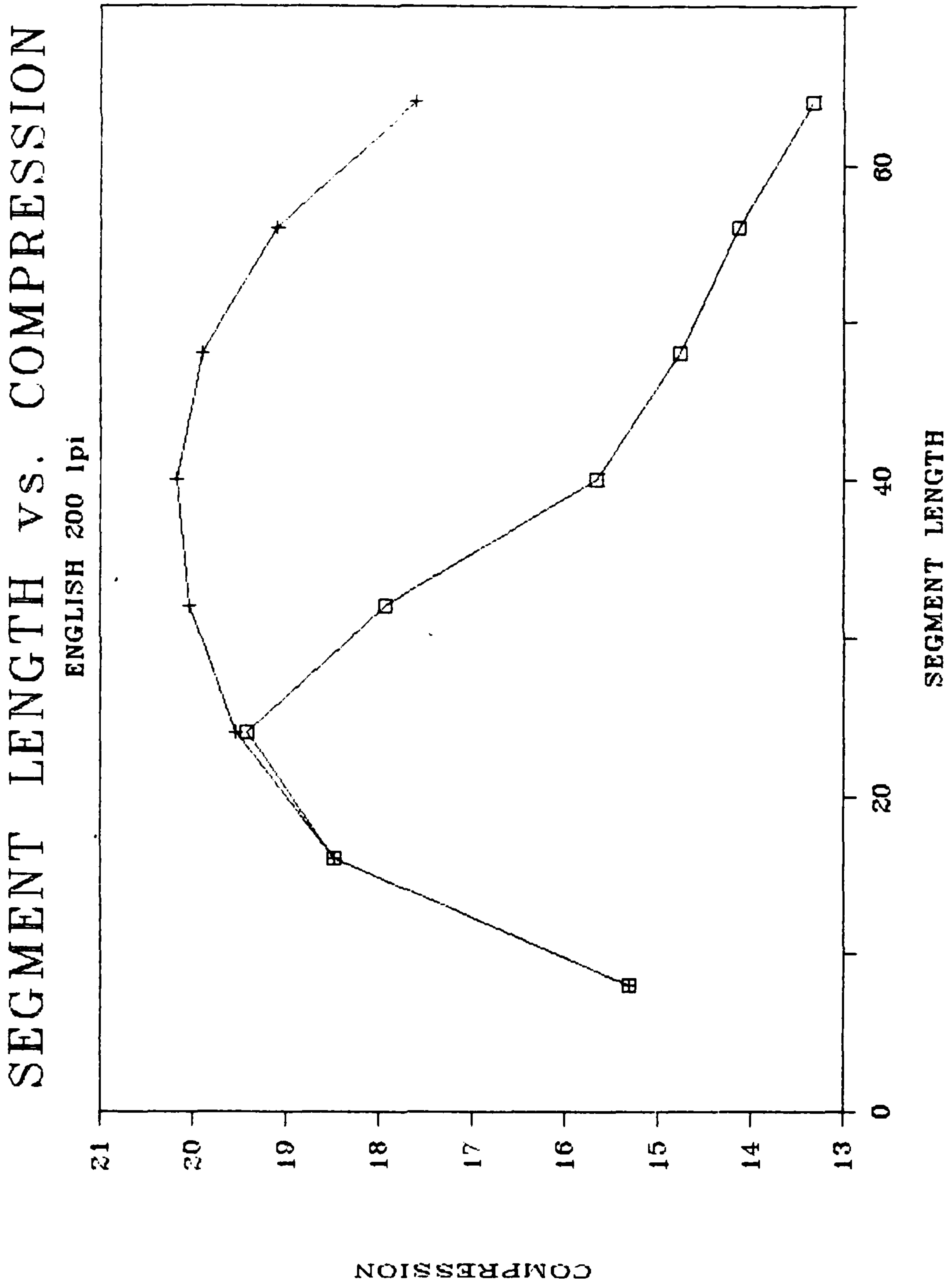

SEGMENT LENGTH VS. COMPRESSION
ENGLISH 200 lpi
SEGMENT LENGTH
COMPRESSION
0
20
40
60
13
14
15
16
17
18
19
20
21

Figure 5.1

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	8.38	17.22
16	16	10.35	18.14
16	24	10.89	15.29
16	32	10.34	14.48
16	40	9.09	14.01
16	48	8.59	13.43
16	56	8.21	13.10
16	64	7.88	12.38
32	24	11.12	24.19
32	32	11.48	30.45
32	40	11.61	21.22
32	48	11.44	20.30
32	56	11.03	20.00
32	64	10.27	19.12

Table 5.2 - Compression Stats, French Journal - 200 LPI

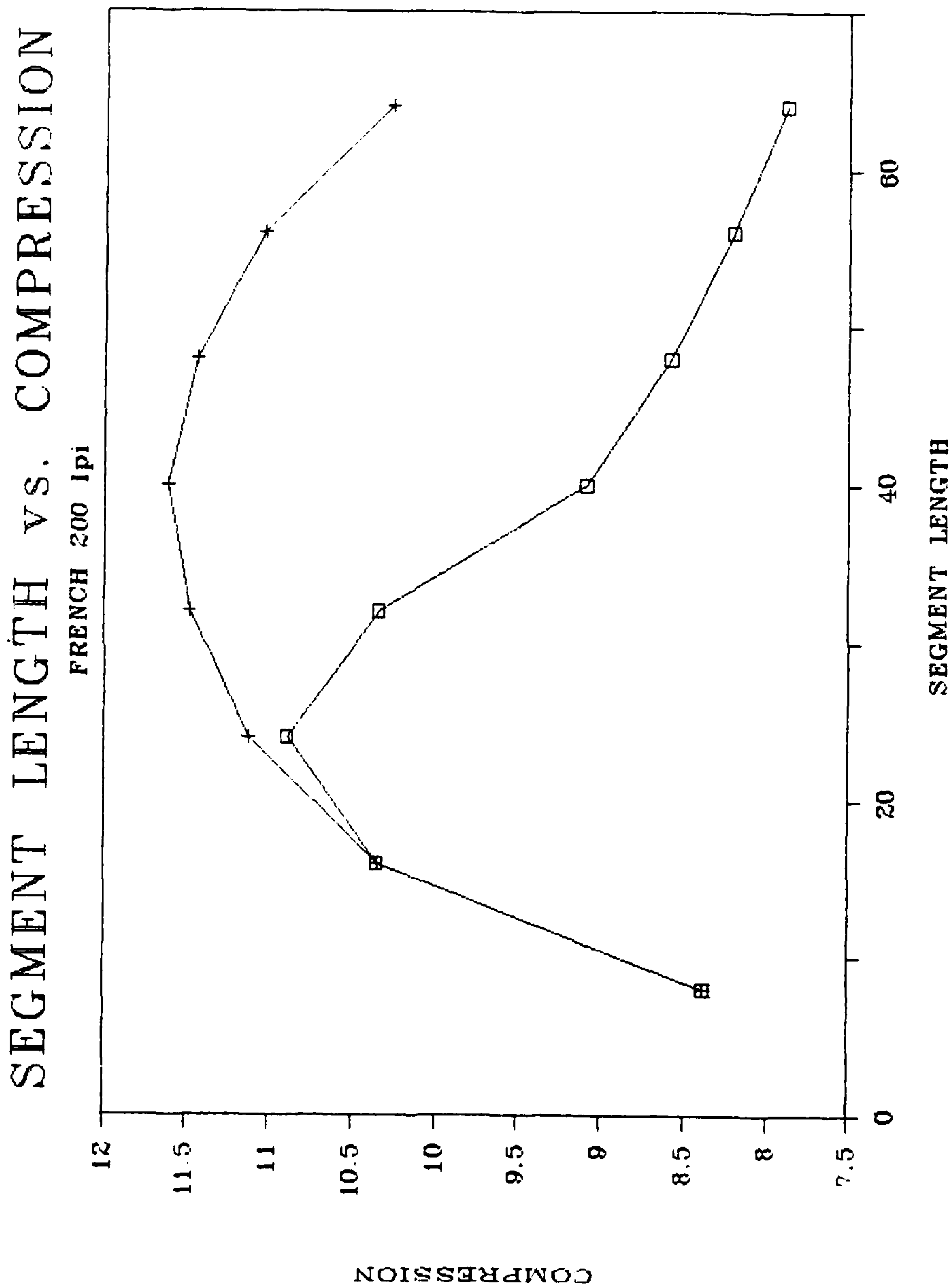

SEGMENT LENGTH VS. COMPRESSION
FRENCH 200 lpi
SEGMENT LENGTH
COMPRESSION
0
20
40
60
12
11.5
11
10.5
10
9.5
9
8.5
8
7.5

Figure 5.2

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	4.88	22.19
16	16	5.41	24.32
16	24	5.33	19.31
16	32	5.00	19.00
16	40	4.06	18.39
16	48	3.68	17.56
16	56	3.40	17.22
16	64	3.15	16.49
32	24	5.41	31.22
32	32	5.44	37.51
32	40	5.44	22.17
32	48	5.41	20.41
32	56	5.37	19.12
32	64	4.99	18.41

Table 5.3 - Compression Stats, Kanji Text - 200 LPI

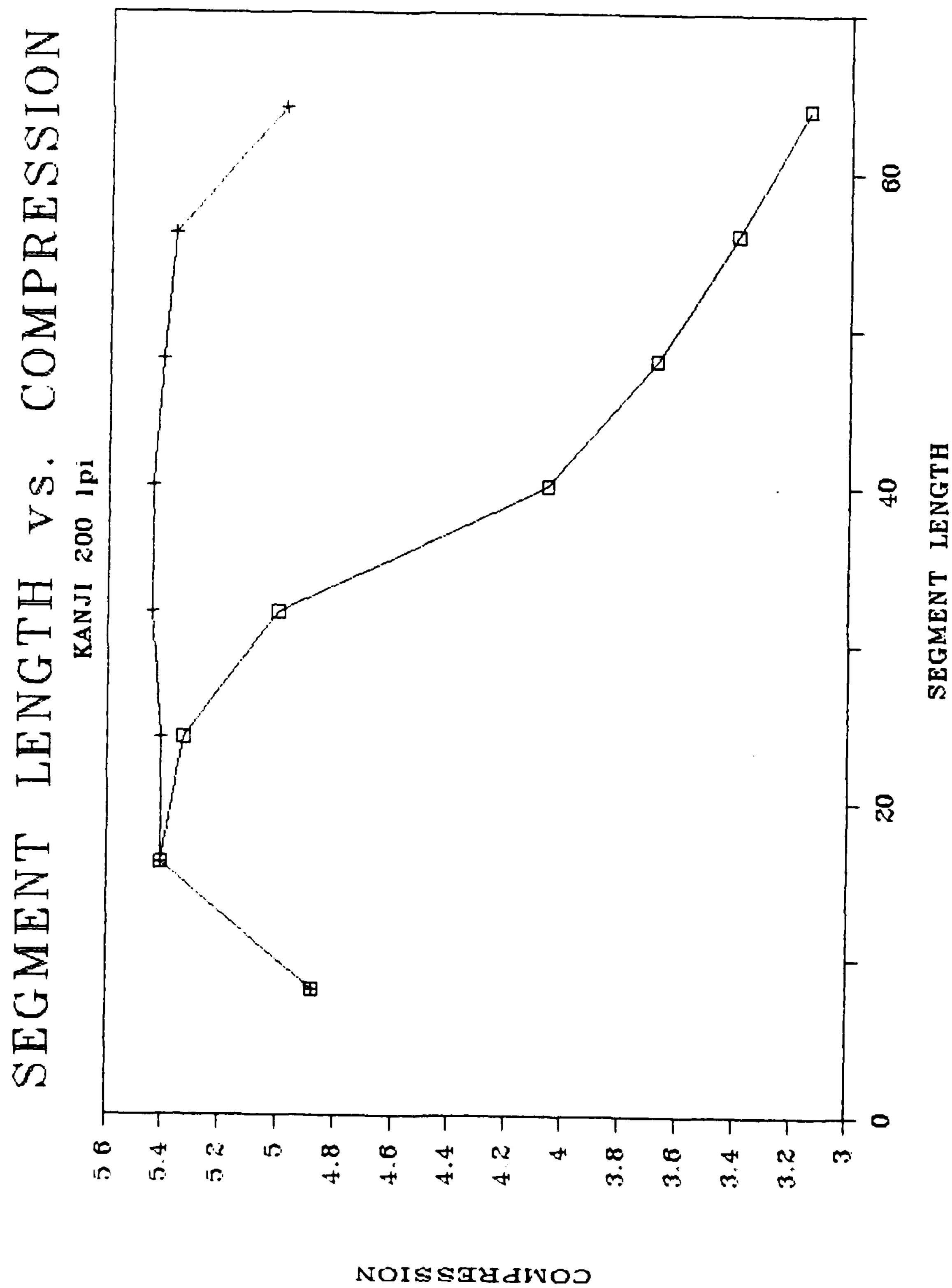

SEGMENT LENGTH VS. COMPRESSION
KANJI 200 lpi
SEGMENT LENGTH
COMPRESSION
0
20
40
60
3
3.2
3.4
3.6
3.8
4
4.2
4.4
4.6
4.8
5
5.2
5.4
5.6

Figure 5.3

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	17.08	13.04
16	16	20.88	14.06
16	24	21.97	10.56
16	32	20.69	9.56
16	40	18.07	9.46
16	48	16.84	9.14
16	56	15.93	9.21
16	64	15.19	9.06
32	24	22.30	17.02
32	32	22.87	20.10
32	40	23.16	12.25
32	48	22.93	11.57
32	56	22.33	11.28
32	64	20.55	10.44

Table 5.4 - Compression Stats, English Letter - 240 LPI

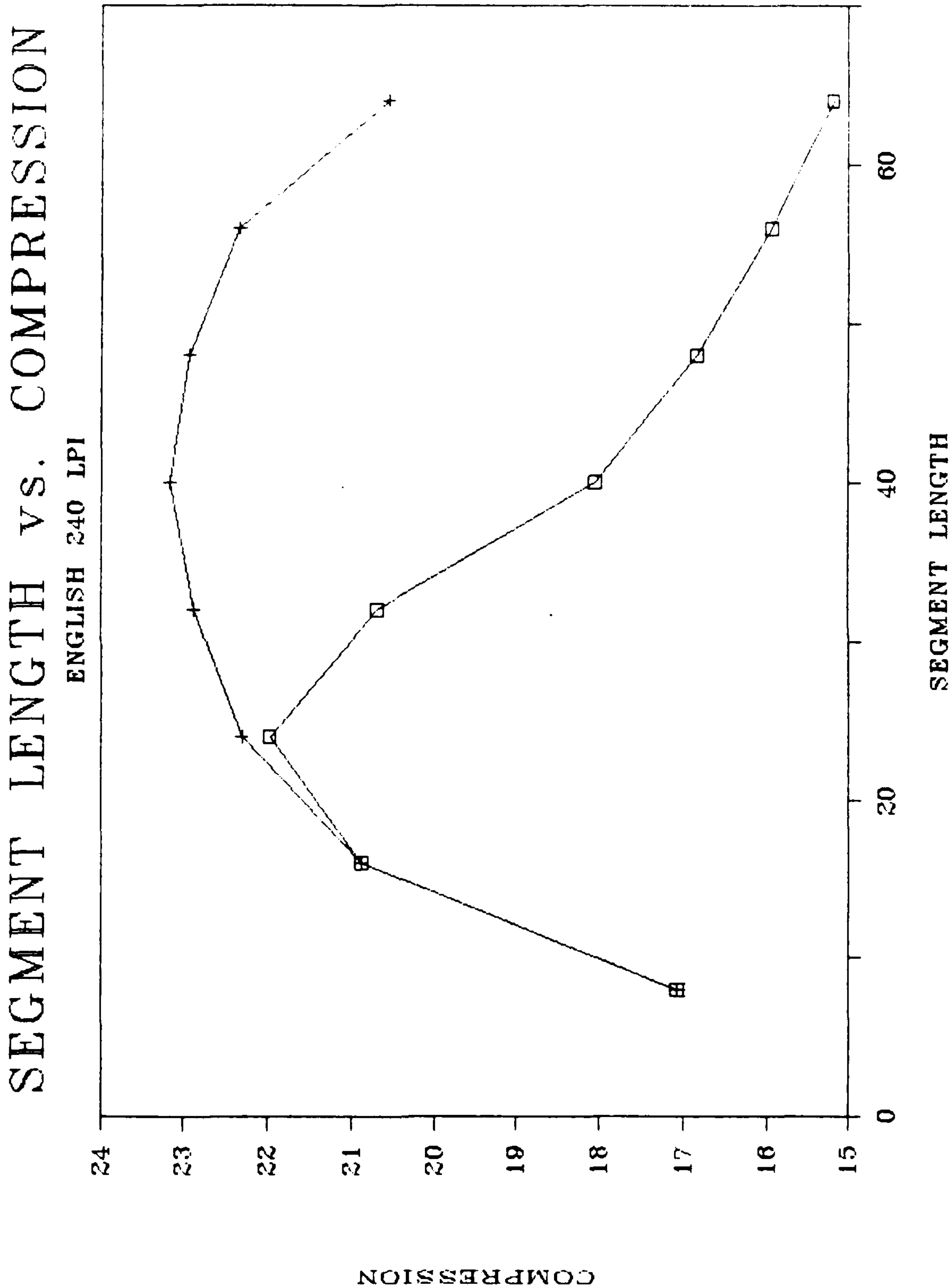

SEGMENT LENGTH VS. COMPRESSION
ENGLISH 240 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
24
23
22
21
20
19
18
17
16
15

Figure 5.4

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	8.77	21.25
16	16	11.25	22.37
16	24	12.01	16.49
16	32	11.52	15.08
16	40	10.16	14.18
16	48	9.62	13.43
16	56	9.16	13.51
16	64	8.70	13.21
32	24	12.21	27.59
32	32	12.69	33.42
32	40	12.90	19.11
32	48	12.83	18.11
32	56	12.52	17.37
32	64	11.65	16.45

Table 5.5 - Compression Stats, French Journal - 240 LPI

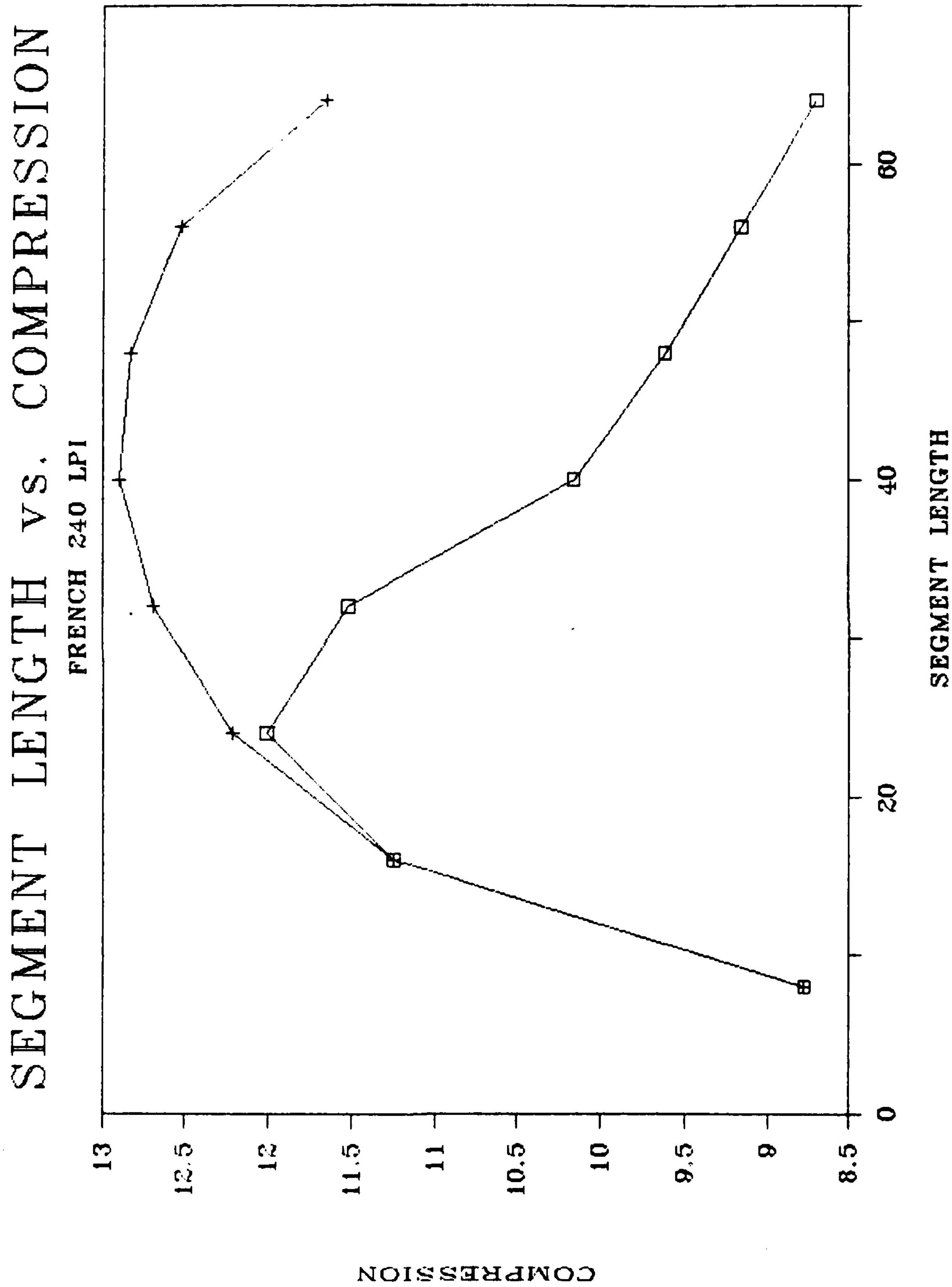

SEGMENT LENGTH VS. COMPRESSION
FRENCH 240 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
13
12.5
12
11.5
11
10.5
10
9.5
9
8.5

Figure 5.5

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	5.33	28.22
16	16	6.18	29.20
16	24	6.11	22.14
16	32	5.75	20.07
16	40	4.78	19.07
16	48	4.30	18.04
16	56	3.96	18.04
16	64	3.63	17.23
32	24	6.18	36.00
32	32	6.20	43.20
32	40	6.23	25.37
32	48	6.20	24.31
32	56	6.22	24.10
32	64	5.84	22.17

Table 5.6 - Compression Stats, Kanji Text - 240 LPI

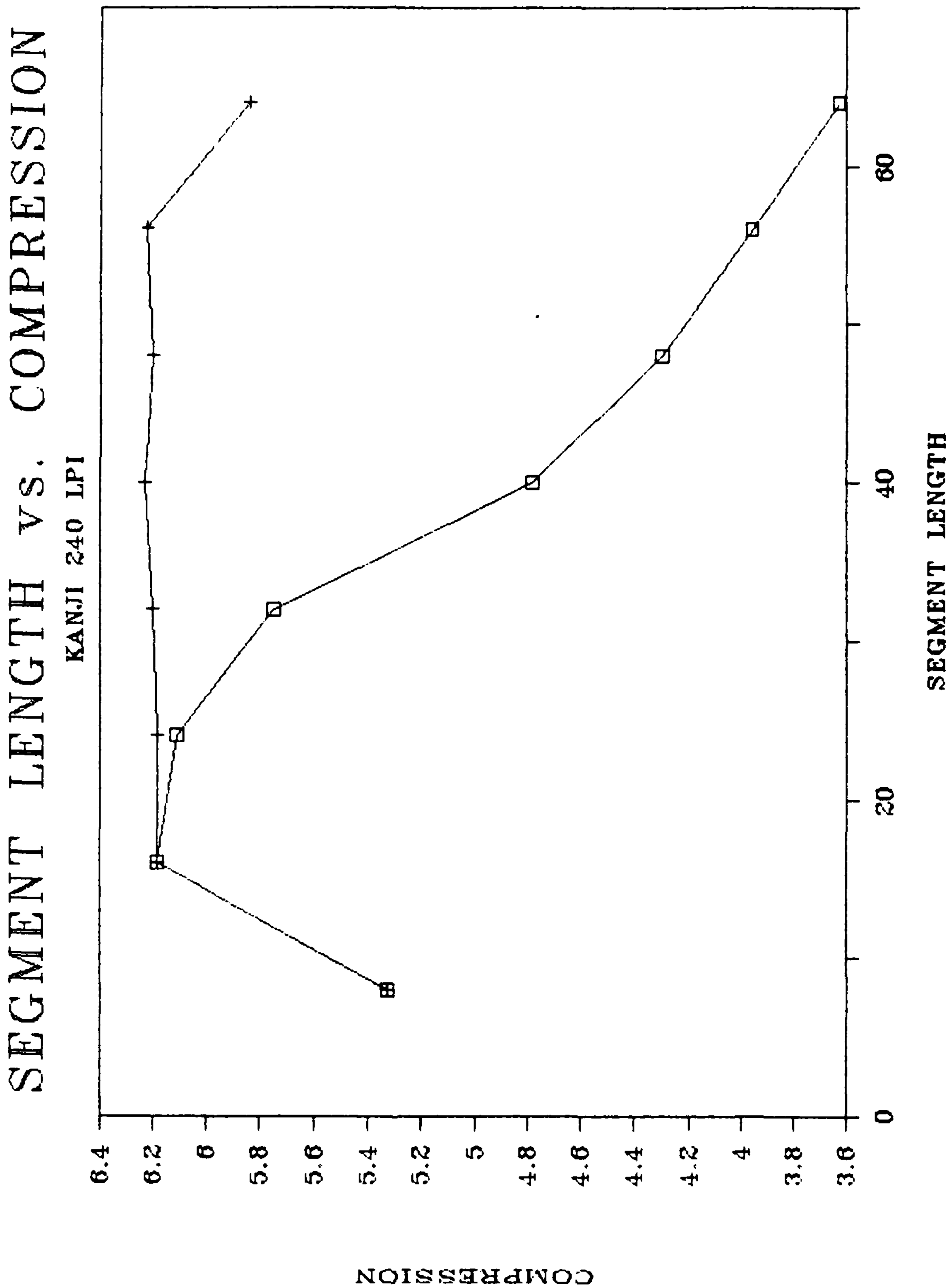

SEGMENT LENGTH VS. COMPRESSION
KANJI 240 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
6.4
6.2
6
5.8
5.6
5.4
5.2
5
4.8
4.6
4.4
4.2
4
3.8
3.6

Figure 5.6

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMRPESSION	RUN TIME
16	8	19.43	18.82
16	16	23.91	20.14
16	24	25.86	16.12
16	32	24.73	15.03
16	40	21.32	14.32
16	48	19.97	14.08
16	56	18.83	14.18
16	64	17.60	13.58
32	24	26.08	23.28
32	32	26.86	27.20
32	40	27.35	18.07
32	48	27.43	17.12
32	56	27.17	16.53
32	64	25.24	16.03

Table 5.7 - Compression Stats, English Letter - 300 LPI

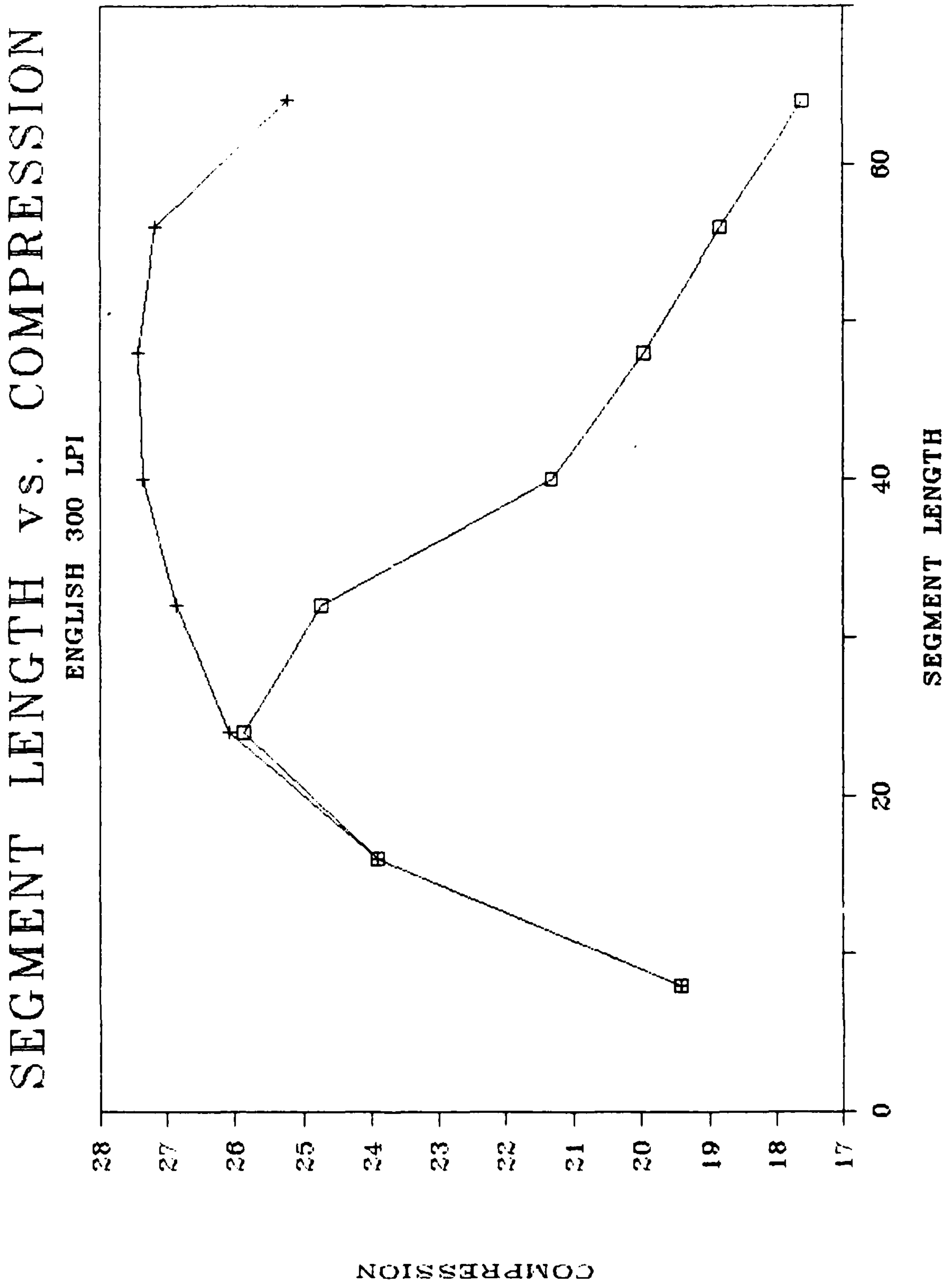

SEGMENT LENGTH VS. COMPRESSION
ENGLISH 300 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
28
27
26
25
24
23
22
21
20
19
18
17

Figure 5.7

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	9.58	35.42
16	16	12.78	34.57
16	24	13.91	32.49
16	32	13.51	27.21
16	40	11.89	26.37
16	48	11.21	25.46
16	56	10.64	25.31
16	64	10.10	24.45
32	24	14.04	41.18
32	32	14.67	45.25
32	40	15.02	38.37
32	48	15.10	37.41
32	56	14.93	36.27
32	64	13.94	35.34

Table 5.8 - Compression Stats, French Journal - 300 LPI

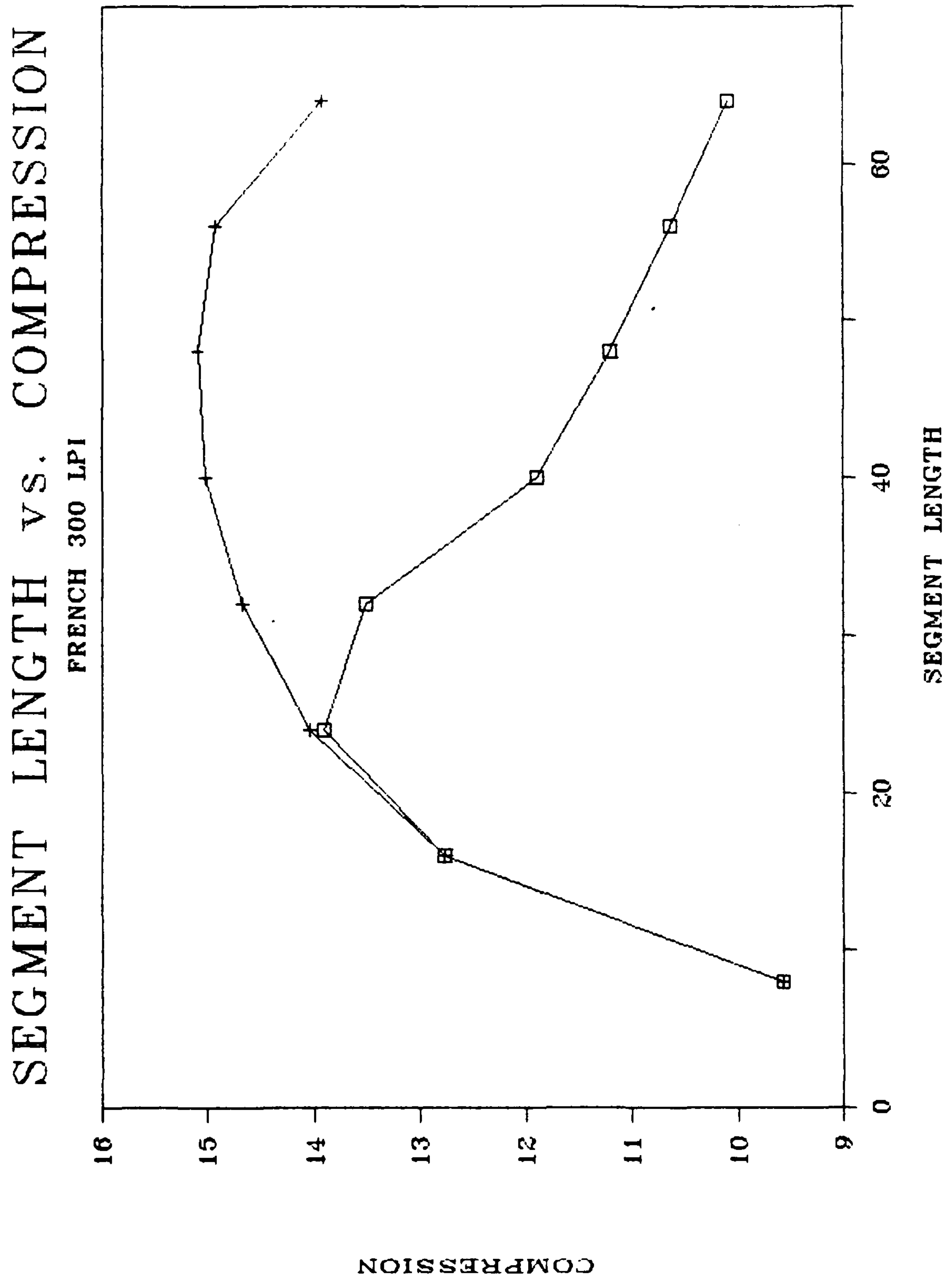

SEGMENT LENGTH VS. COMPRESSION
FRENCH 300 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
9
10
11
12
13
14
15
16

Figure 5.8

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	5.91	40.57
16	16	7.17	41.09
16	24	7.29	32.21
16	32	6.86	29.40
16	40	5.83	29.34
16	48	5.33	27.55
16	56	4.90	28.01
16	64	4.43	26.22
32	24	7.35	49.36
32	32	7.33	57.34
32	40	7.34	36.03
32	48	7.32	34.07
32	56	7.40	34.04
32	64	7.18	32.41

Table 5.9 - Compression Stats, Kanji Text - 300 LPI

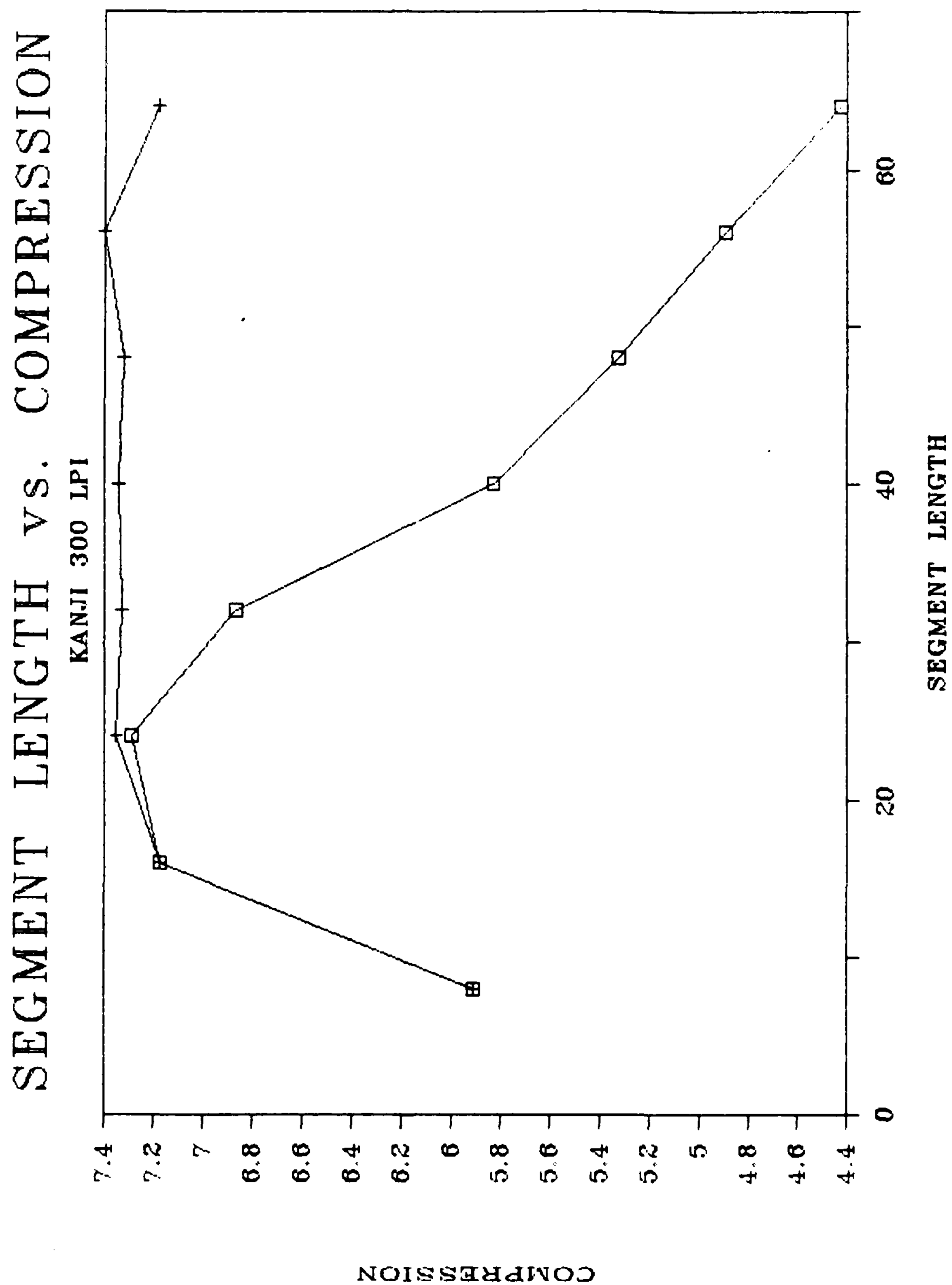
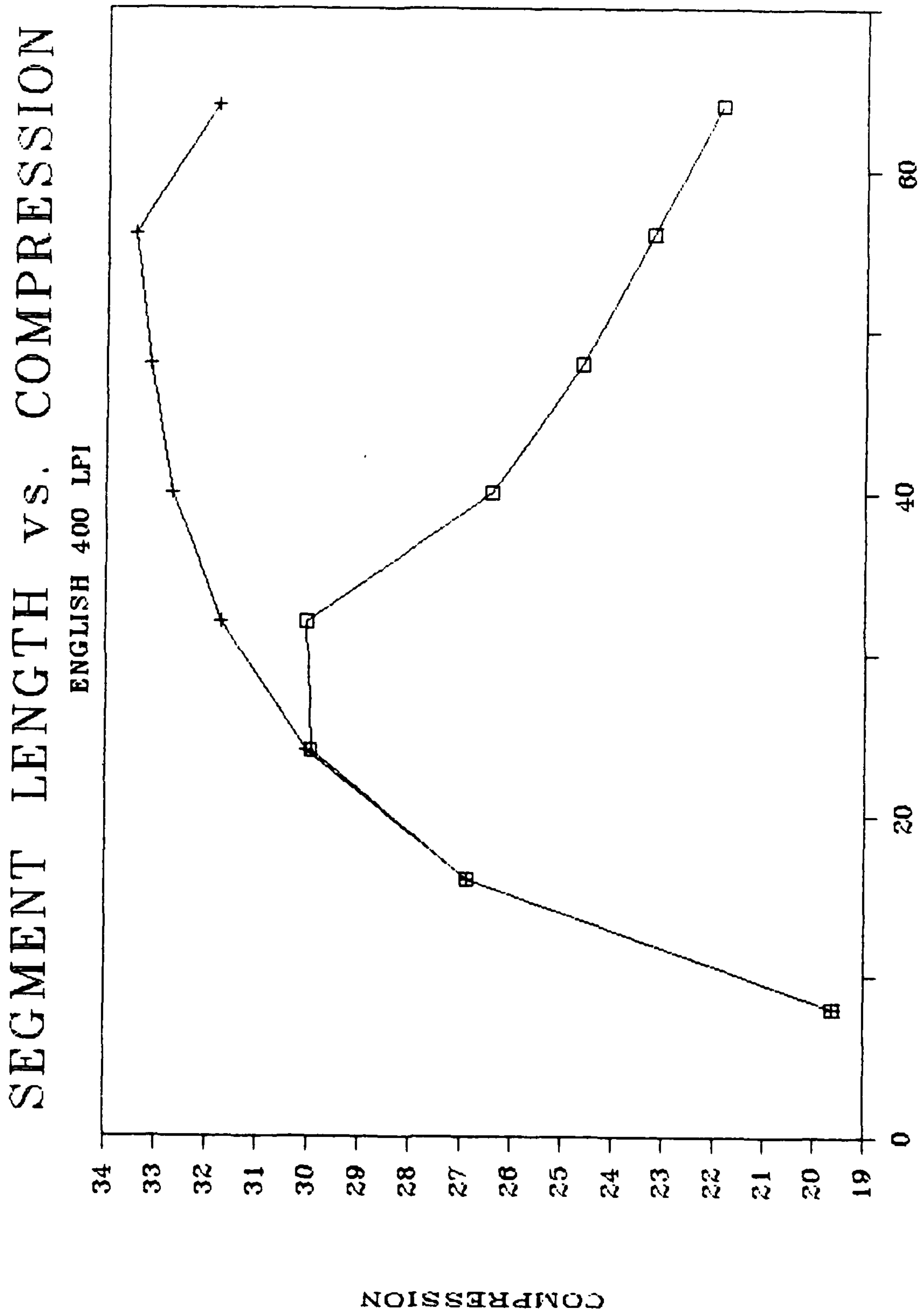

SEGMENT LENGTH VS. COMPRESSION
KANJI 300 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
7.4
7.2
7
6.8
6.6
6.4
6.2
6
5.8
5.6
5.4
5.2
5
4.8
4.6
4.4

Figure 5.9

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	19.63	35.39
16	16	26.88	35.05
16	24	29.95	29.11
16	32	30.04	27.24
16	40	26.40	26.37
16	48	24.63	25.40
16	56	23.24	26.00
16	64	21.88	25.23
32	24	30.04	39.48
32	32	31.71	45.02
32	40	32.66	31.15
32	48	33.11	30.01
32	56	33.45	29.52
32	64	31.81	28.31

Table 5.10 - Compression Stats, English Letter - 400 LPI



SEGMENT LENGTH VS. COMPRESSION
ENGLISH 400 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19

Figure 5.10

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	10.40	54.47
16	16	14.67	53.14
16	24	16.62	48.23
16	32	16.67	40.23
16	40	14.75	37.45
16	48	13.94	36.27
16	56	13.25	36.40
16	64	12.45	35.25
32	24	16.68	1:01.34
32	32	17.65	1:08.29
32	40	18.31	45.50
32	48	18.62	43.57
32	56	18.81	42.28
32	64	16.89	40.01

Table 5.11 - Compression Stats, French Journal - 400 LPI

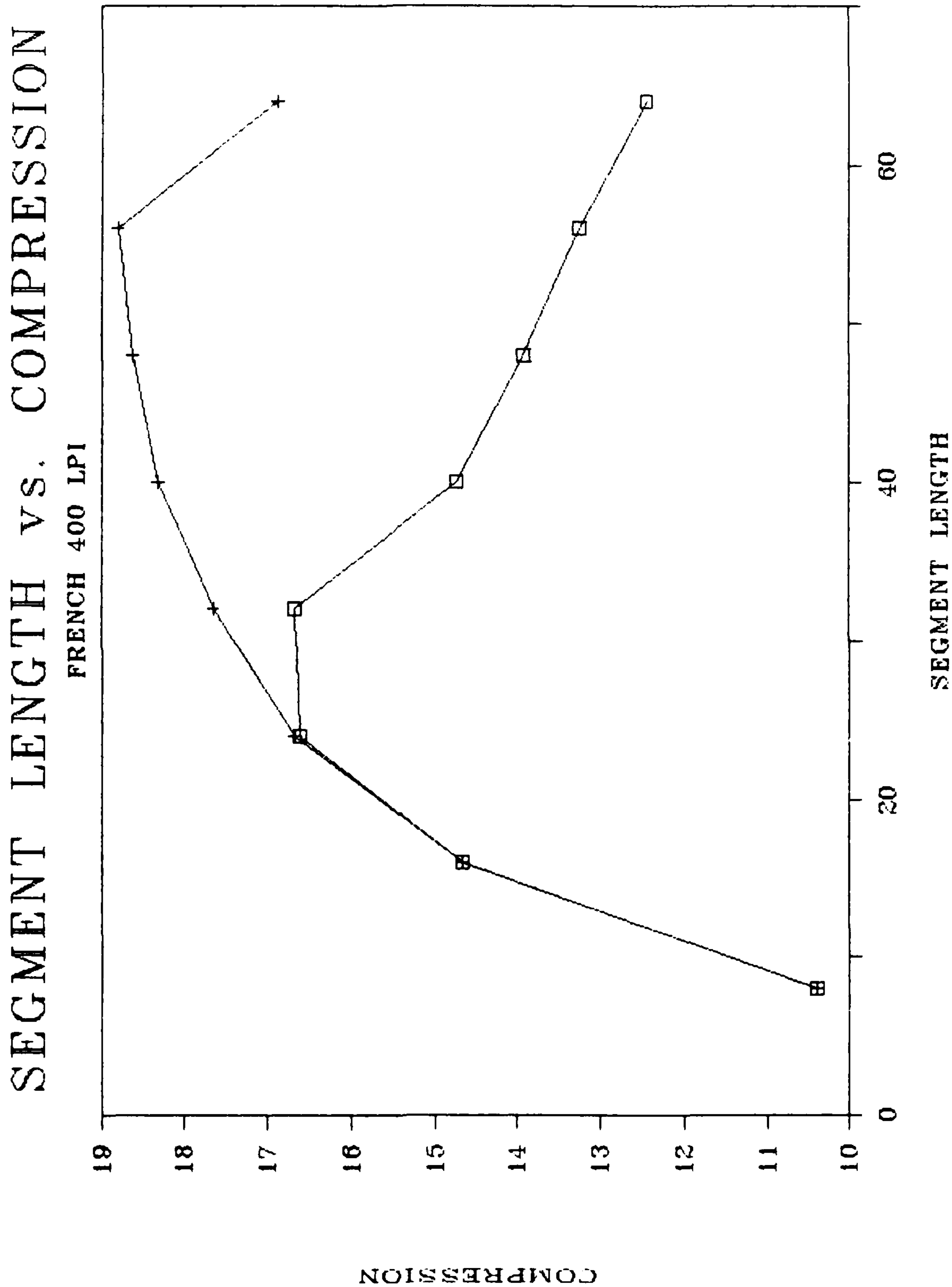

SEGMENT LENGTH VS. COMPRESSION
FRENCH 400 LPI
SEGMENT LENGTH
COMPRESSION
0
20
40
60
10
11
12
13
14
15
16
17
18
19

Figure 5.11

ARITHMETIC WORD LENGTH	SEGMENT LENGTH	COMPRESSION	RUN TIME
16	8	6.71	1:09:46
16	16	8.63	1:06.20
16	24	9.26	55.22
16	32	8.89	49.39
16	40	7.58	47.53
16	48	7.00	46.03
16	56	6.59	47.24
16	64	6.05	45.11
32	24	9.32	1:16.32
32	32	9.39	1:26.23
32	40	9.32	58.29
32	48	9.28	55.56
32	56	9.40	56.01
32	64	9.12	53.27

Table 5.12 - Compression Stats, Kanji Text - 400 LPI

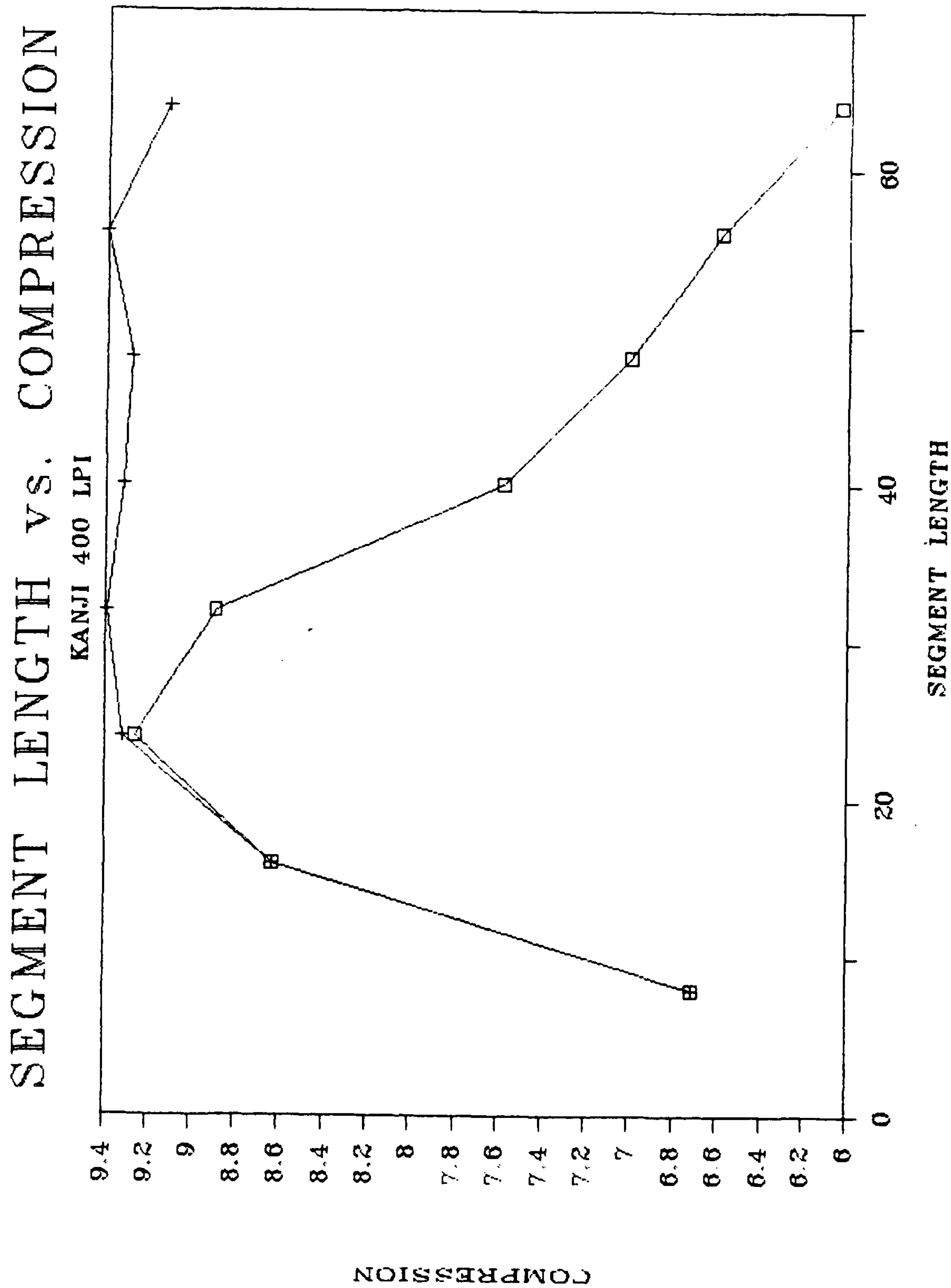

SEGMENT LENGTH VS. COMPRESSION
KANJI 400 LPI
SEGMENT LENGTH
0
20
40
60
COMPRESSION
9.4
9.2
9
8.8
8.6
8.4
8.2
8
7.8
7.6
7.4
7.2
7
6.8
6.6
6.4
6.2
6

Figure 5.12

Table 5.13 - Suggested Parameters for SLDC Encoding Program

Suggested Segment Lengths for Optimum SLDC Compression

File	16 bit Arithmetic word length	32 bit Arithmetic word length
English 200 lpi	24	40
French 200 lpi	24	40
Kanji 200 lpi	16	40
English 240 lpi	24	40
French 240 lpi	24	40
Kanji 240 lpi	16	40
English 300 lpi	24	48
French 300 lpi	24	48
Kanji 300 lpi	24	56
English 400 lpi	32	56
French 400 lpi	32	56
Kanji 400 lpi	24	56

An analysis of the 32 and 64 bit histograms described in the SOW versus those expected on the basis of a memoryless binomial distribution was made in order to determine if the deviation between the actual and expected probability distributions was significant. The results of this analysis are presented graphically in Figures 5.13 through 5.33 (In the probability distribution graphs, the □ indicates the expected distribution curve and the + indicates the actual distribution curve.). Every graph but the 200 resolution Kanji and 64 bit 400 resolution Kanji graphs have a second graph associated with it in which the deviation between the curves is presented in greater detail. The results show definite significant deviation between the actual and expected histograms, and is most evident in the Kanji graphs. It appears that the number of occurrences in both 32 and 64 bit cases of segment weights < 3 are significantly lower than expected, and occurrences of segment weights > 3 are significantly higher than expected. Attempts were made to alleviate this problem by trying to produce a more predictable CIF by increasing the neighbor template from 4 to 7 bits. The three extra bits were placed in different positions available to the decoder, around the already existing template. This attempt made the actual curve slightly more binomial in nature, but changes in compression were small.

There are two possible ways to improve the SLDC technique. The first is the aforementioned attempt to improve the CIF to be more binomially predictable, which was not successful. The second would be to replace the predictive encoding algorithm, which is binomially based, with a function which better fits the actual statistics of the CIF's.

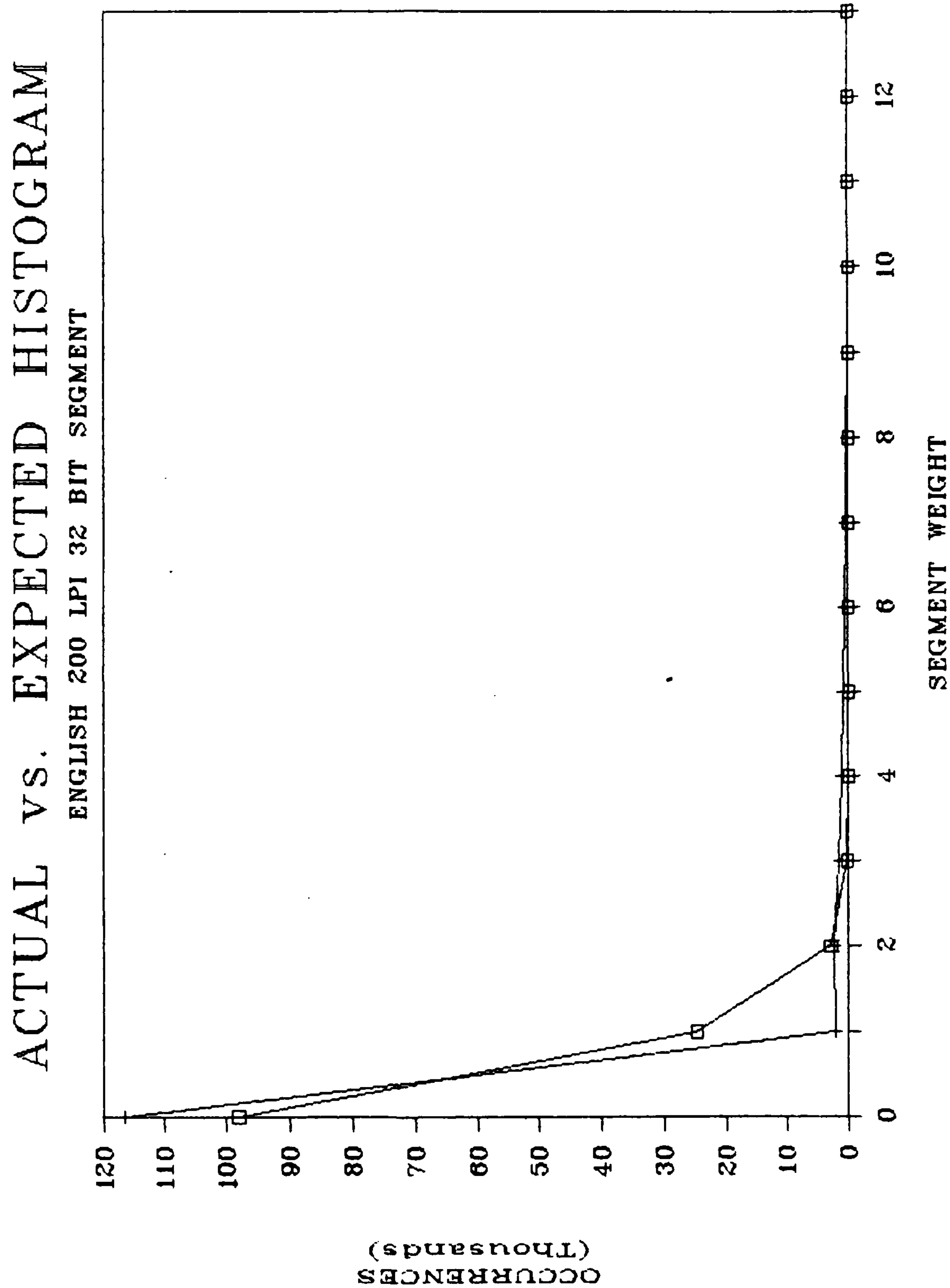


Figure 5.13 - Full Scale

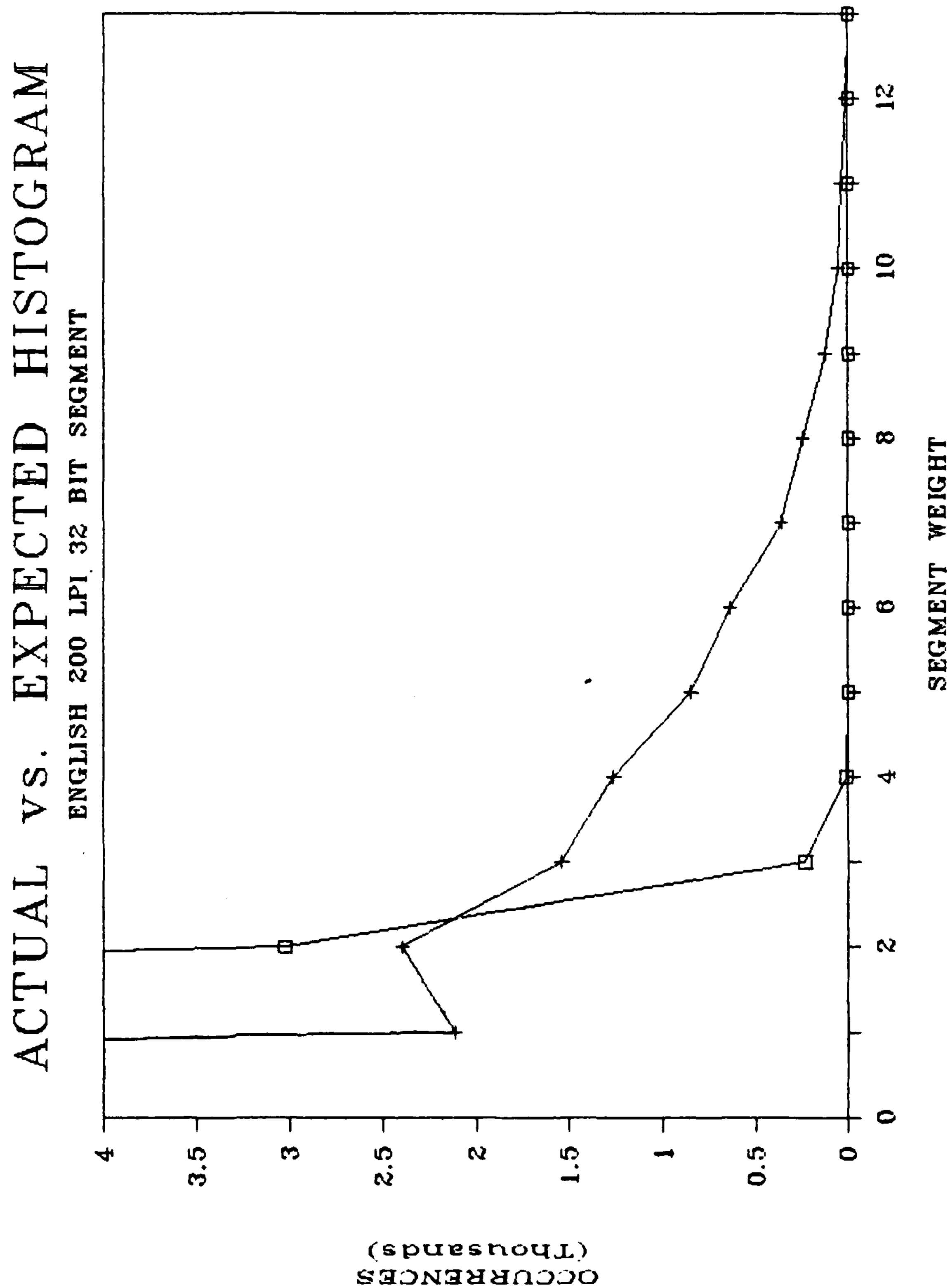


Figure 5.14 - Enlarged Section

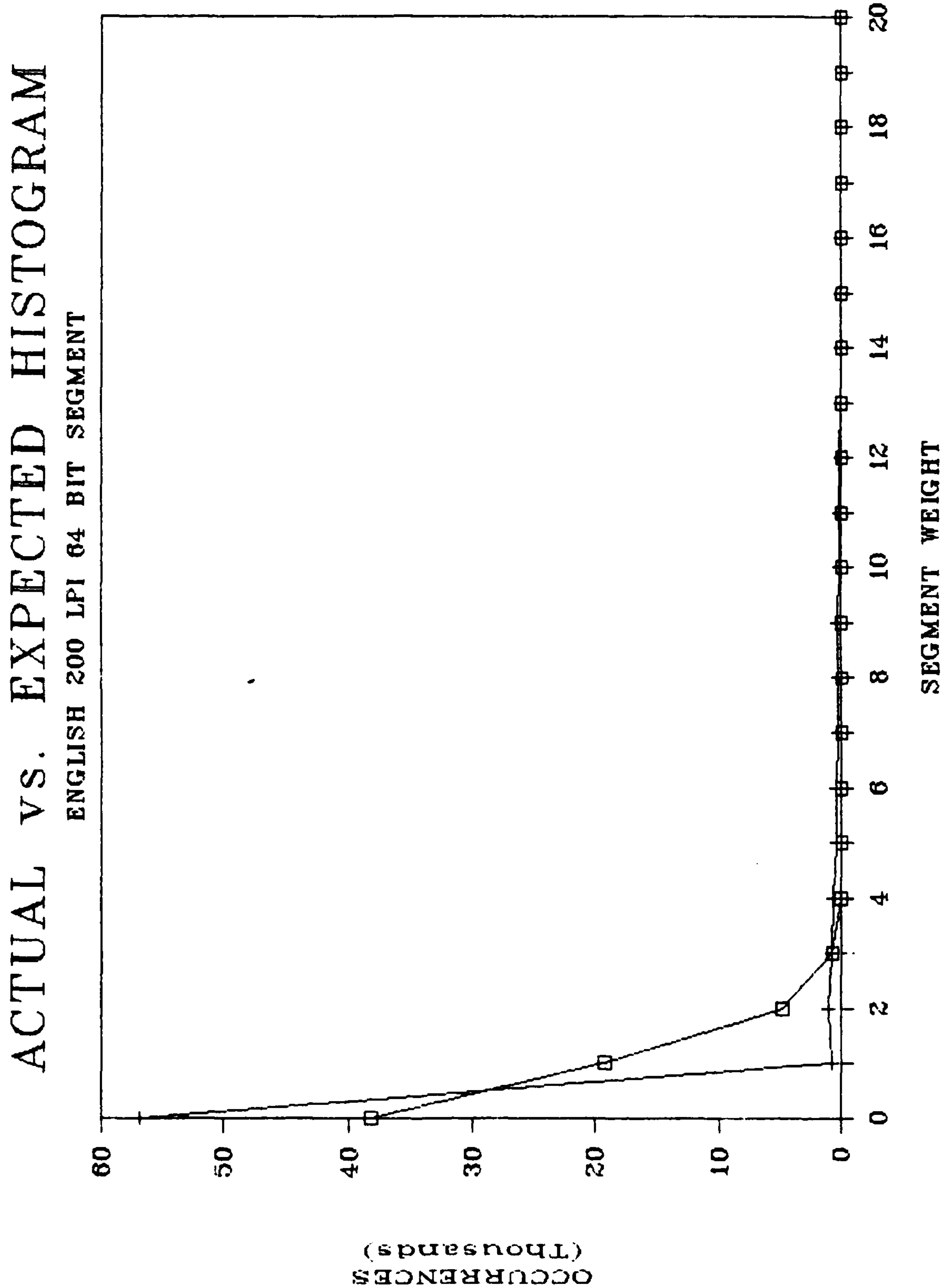


Figure 5.15 - Full Scale

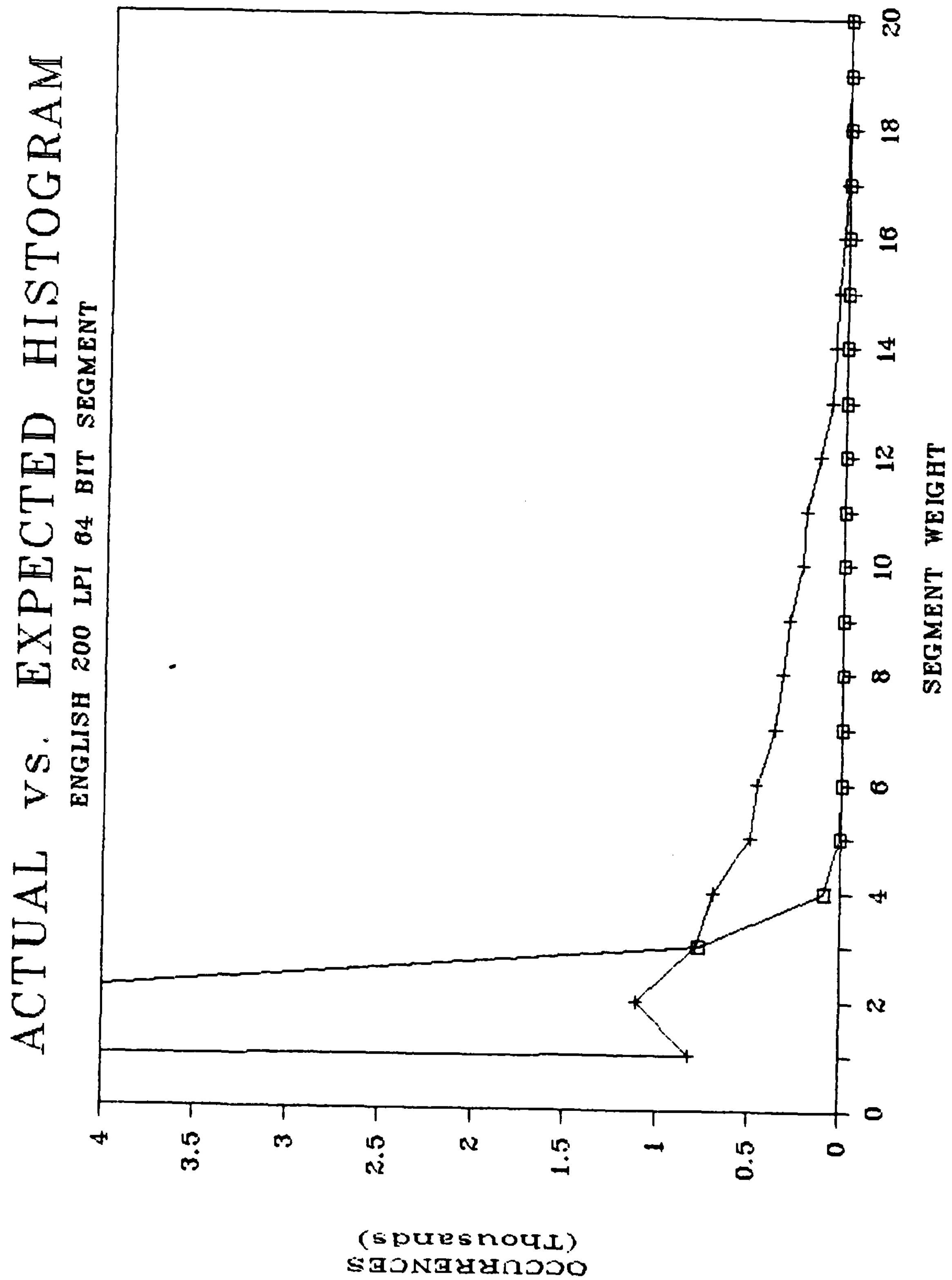


Figure 5.16 - Enlarged Section

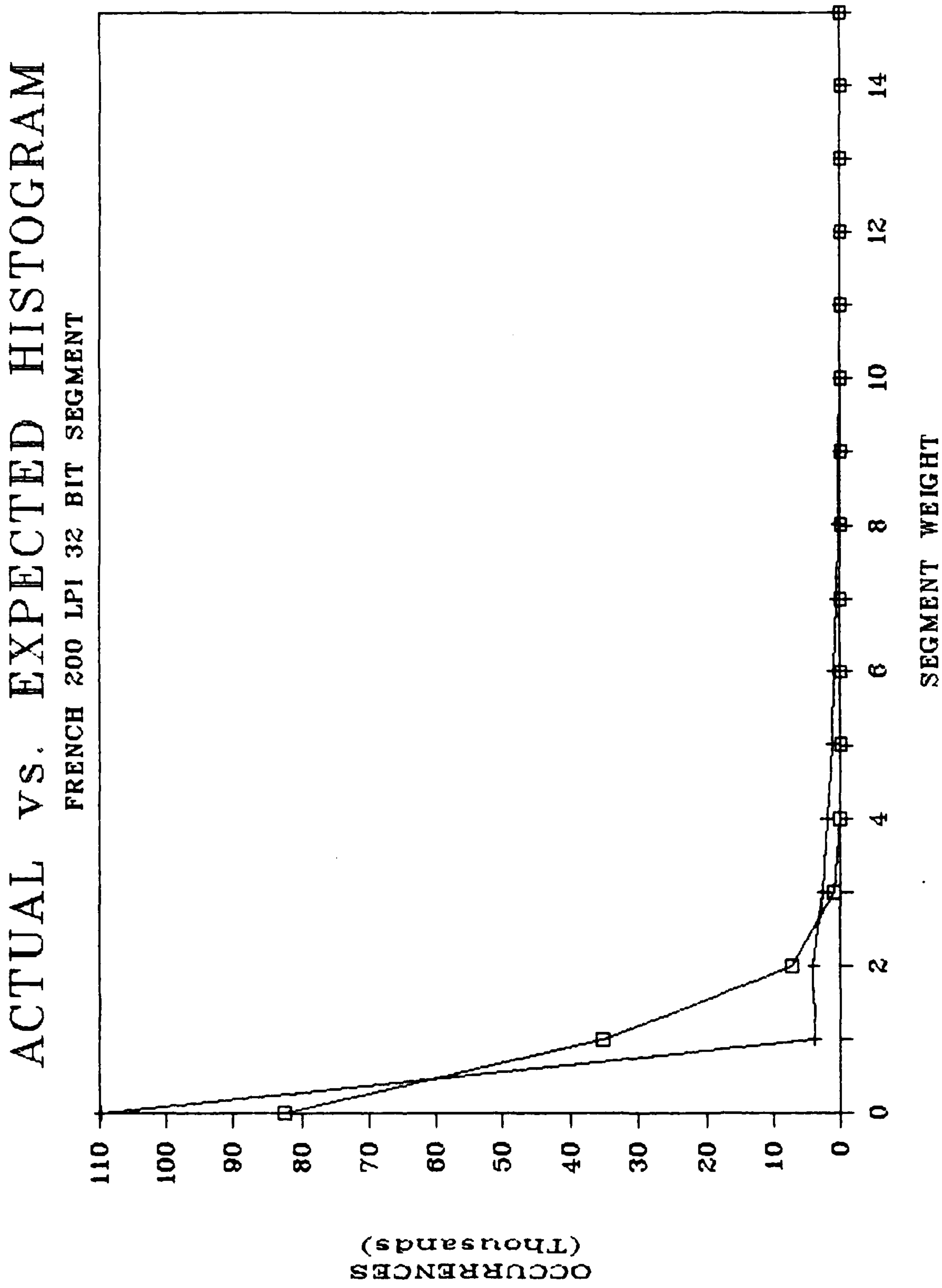


Figure 5.17 - Full Scale

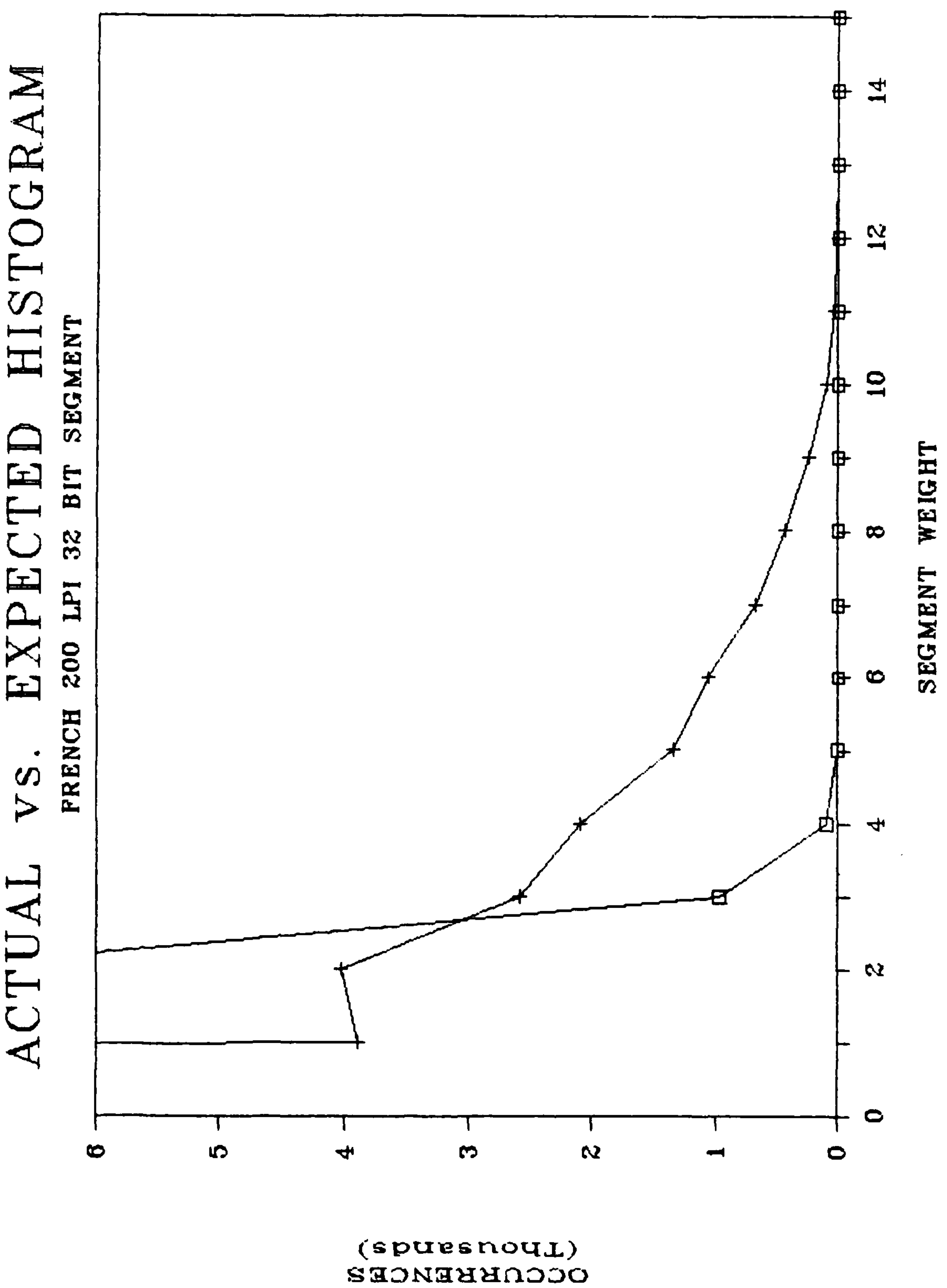


Figure 5.18 - Enlarged Section

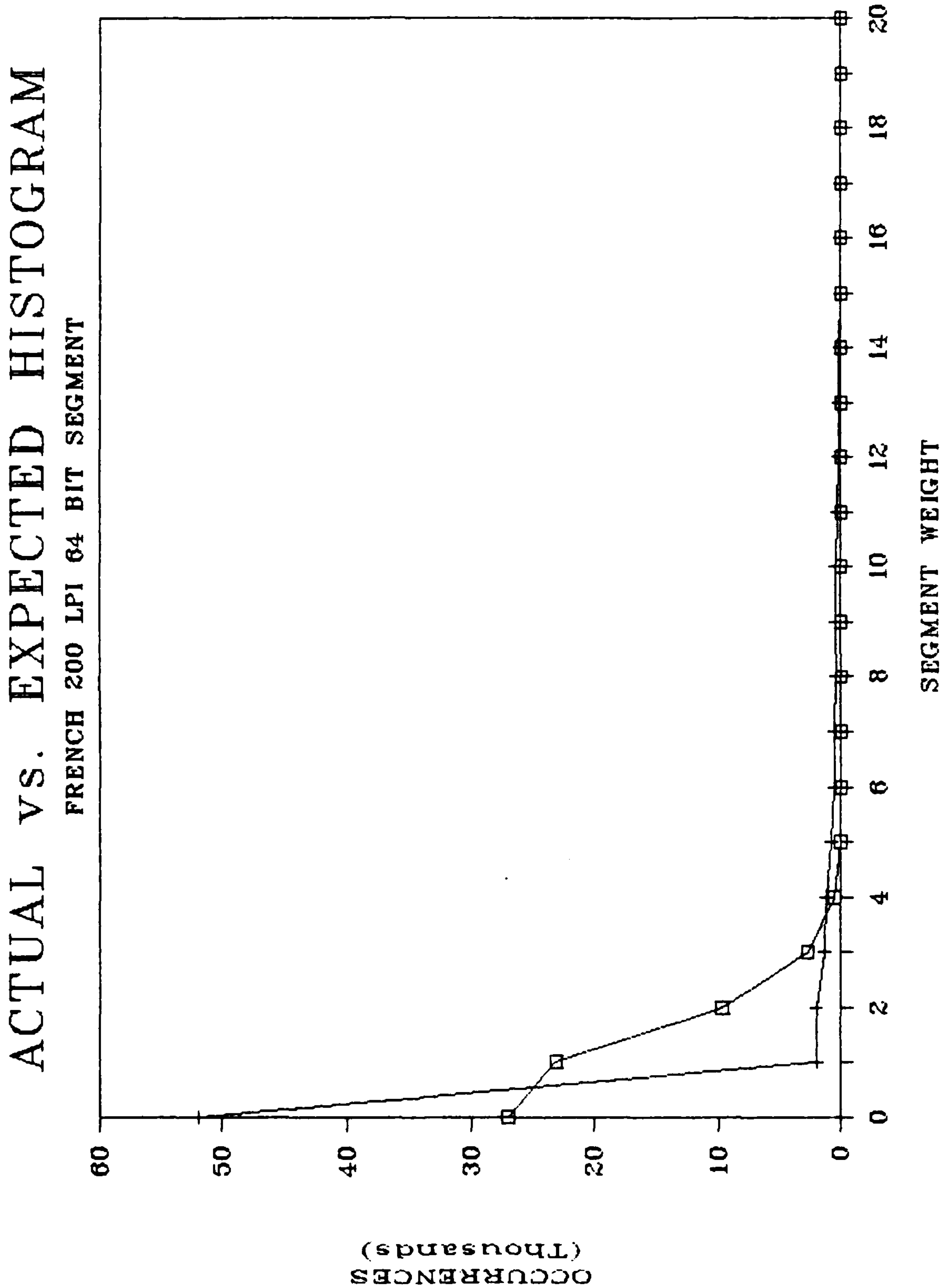


Figure 5.19 - Full Scale

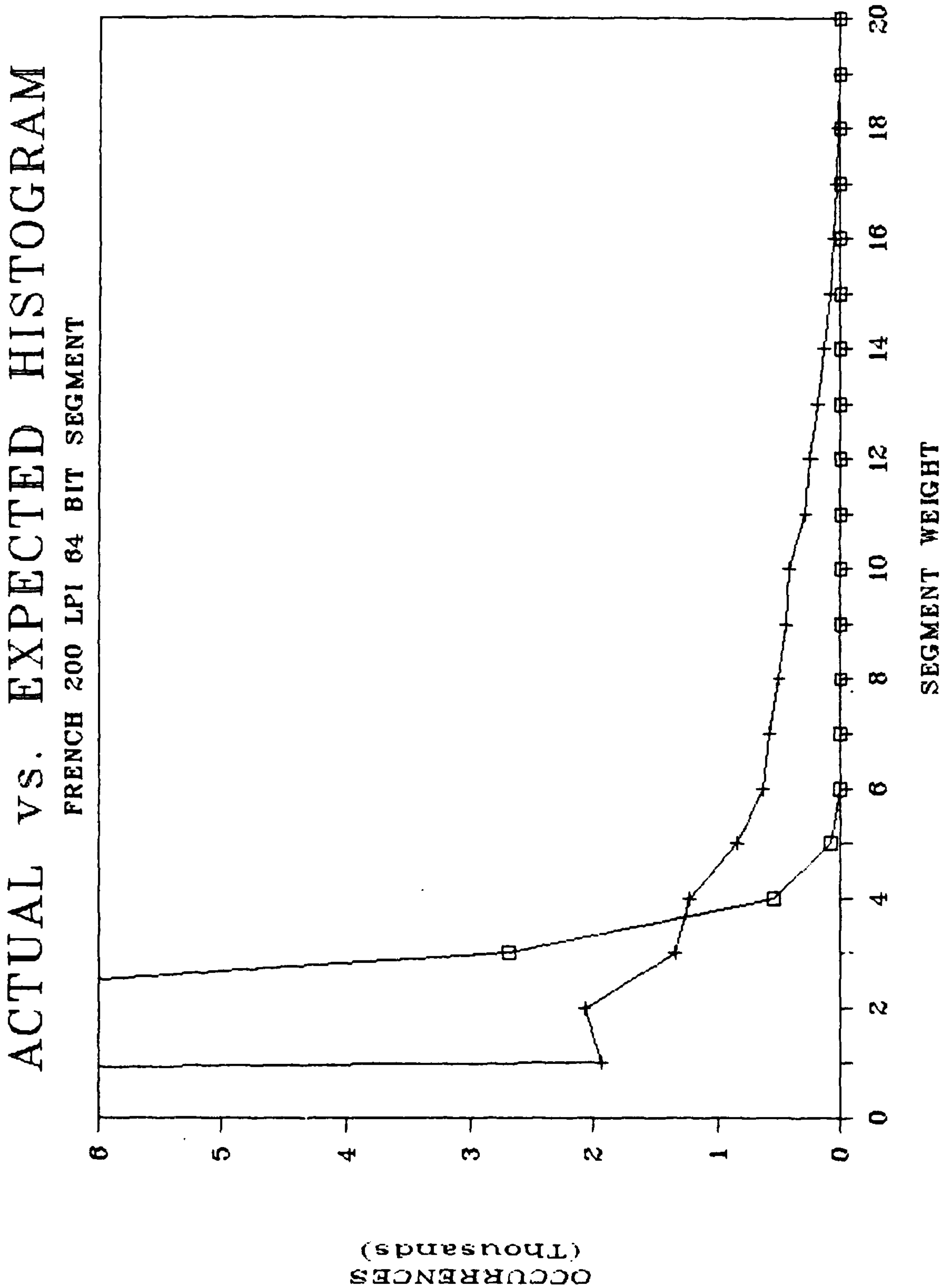


Figure 5.20 - Enlarged Section

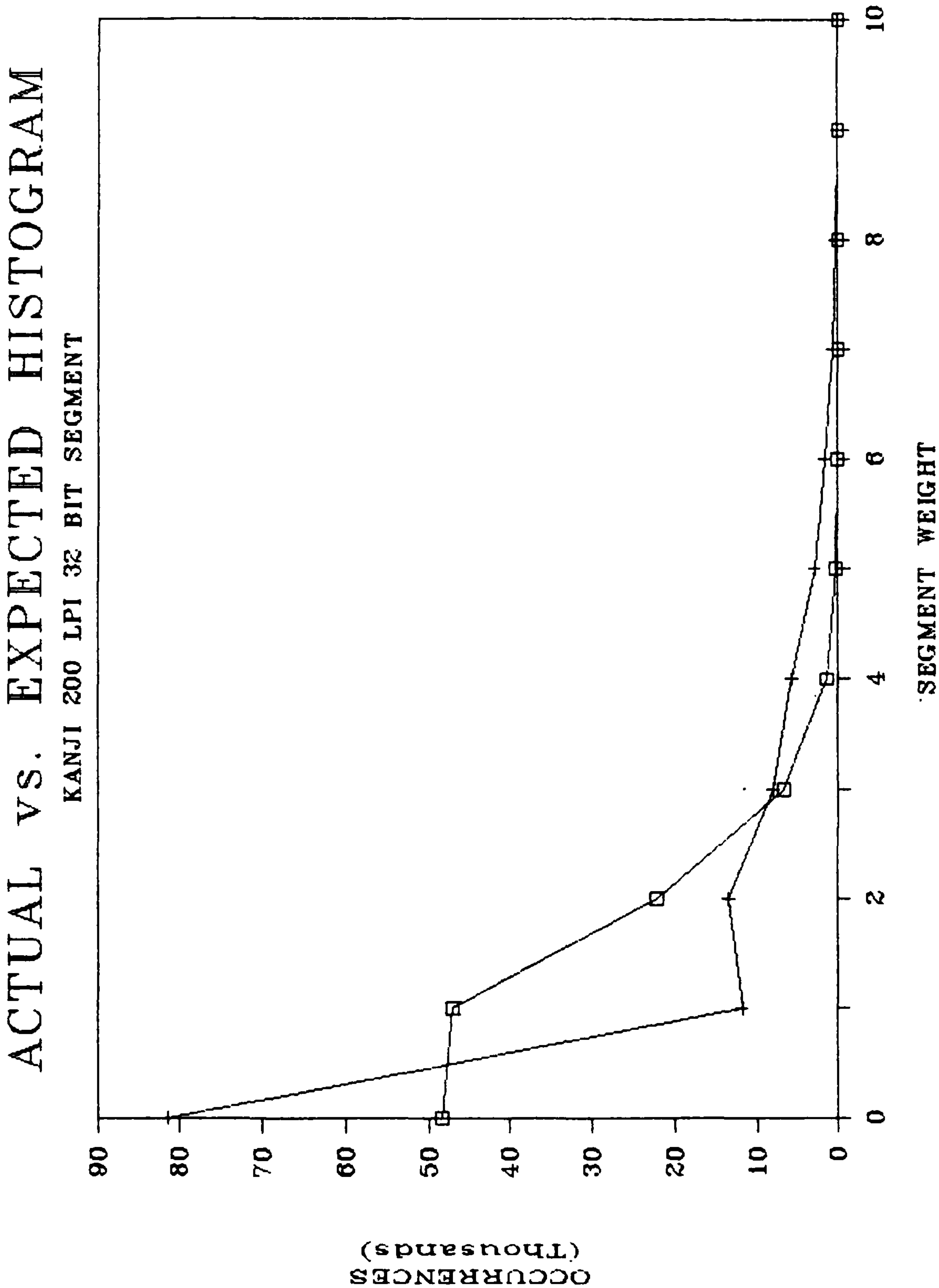


Figure 5.21 - Full Scale

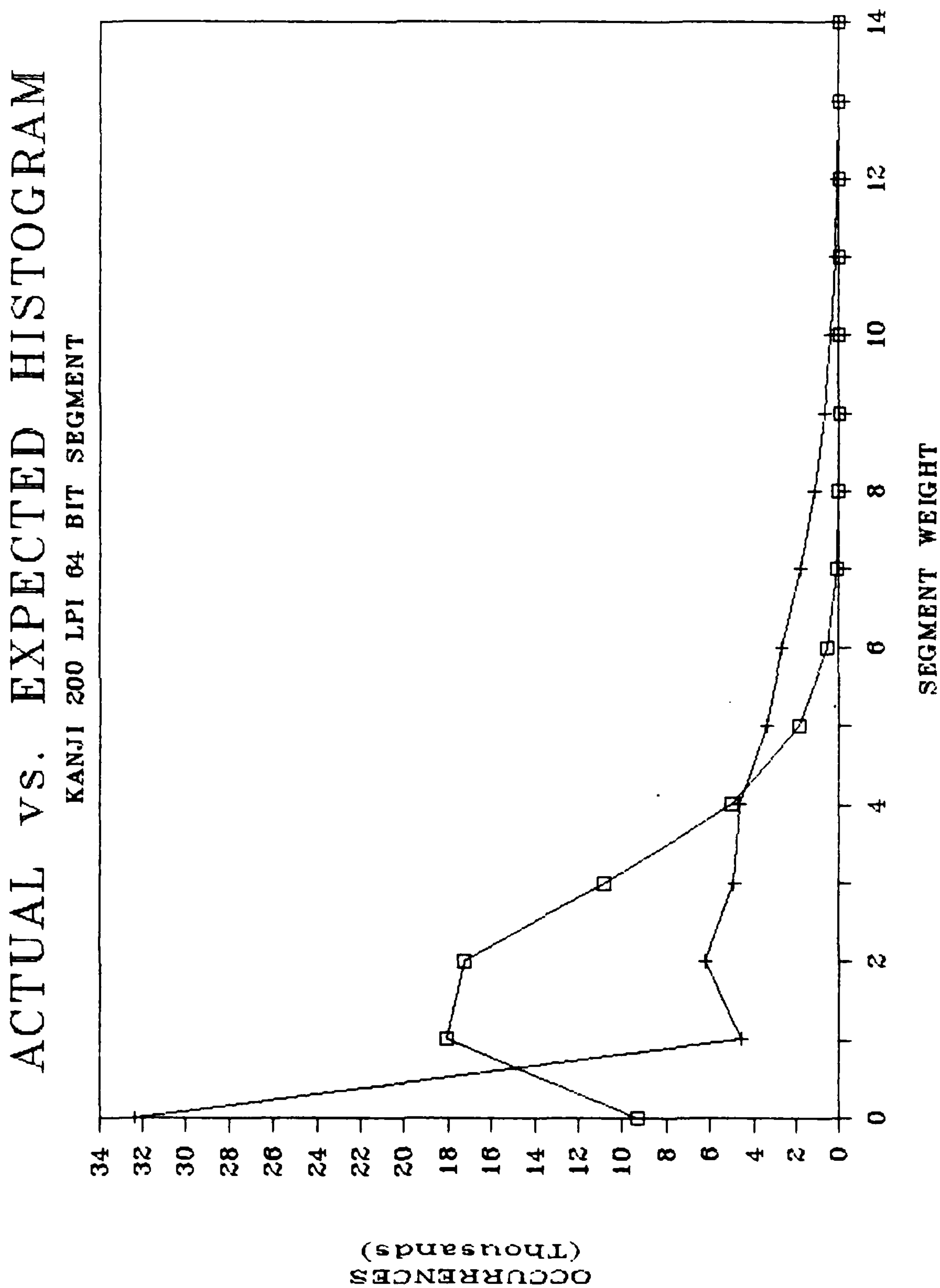


Figure 5.22 - Full Scale

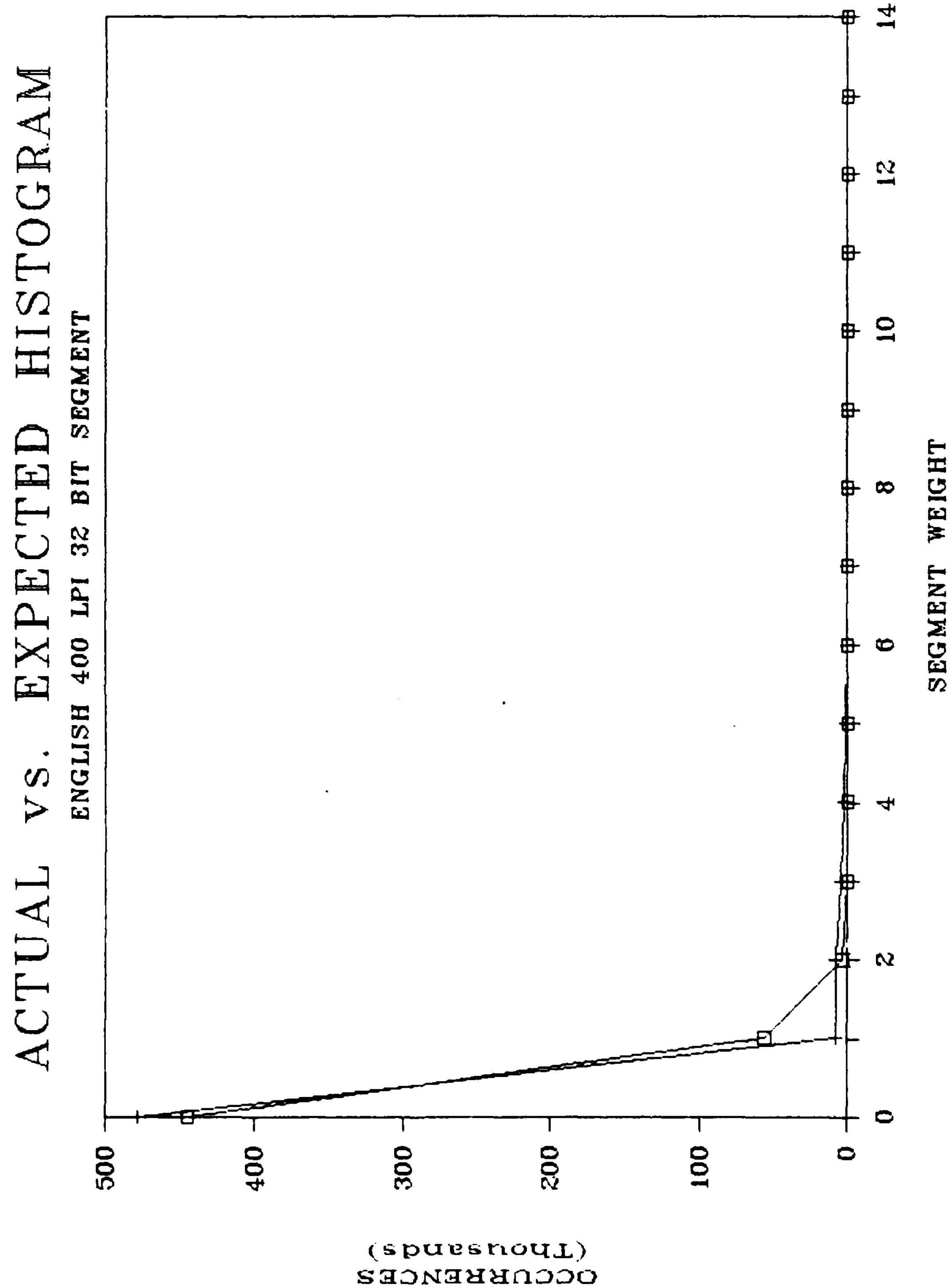


Figure 5.23 - Full Scale

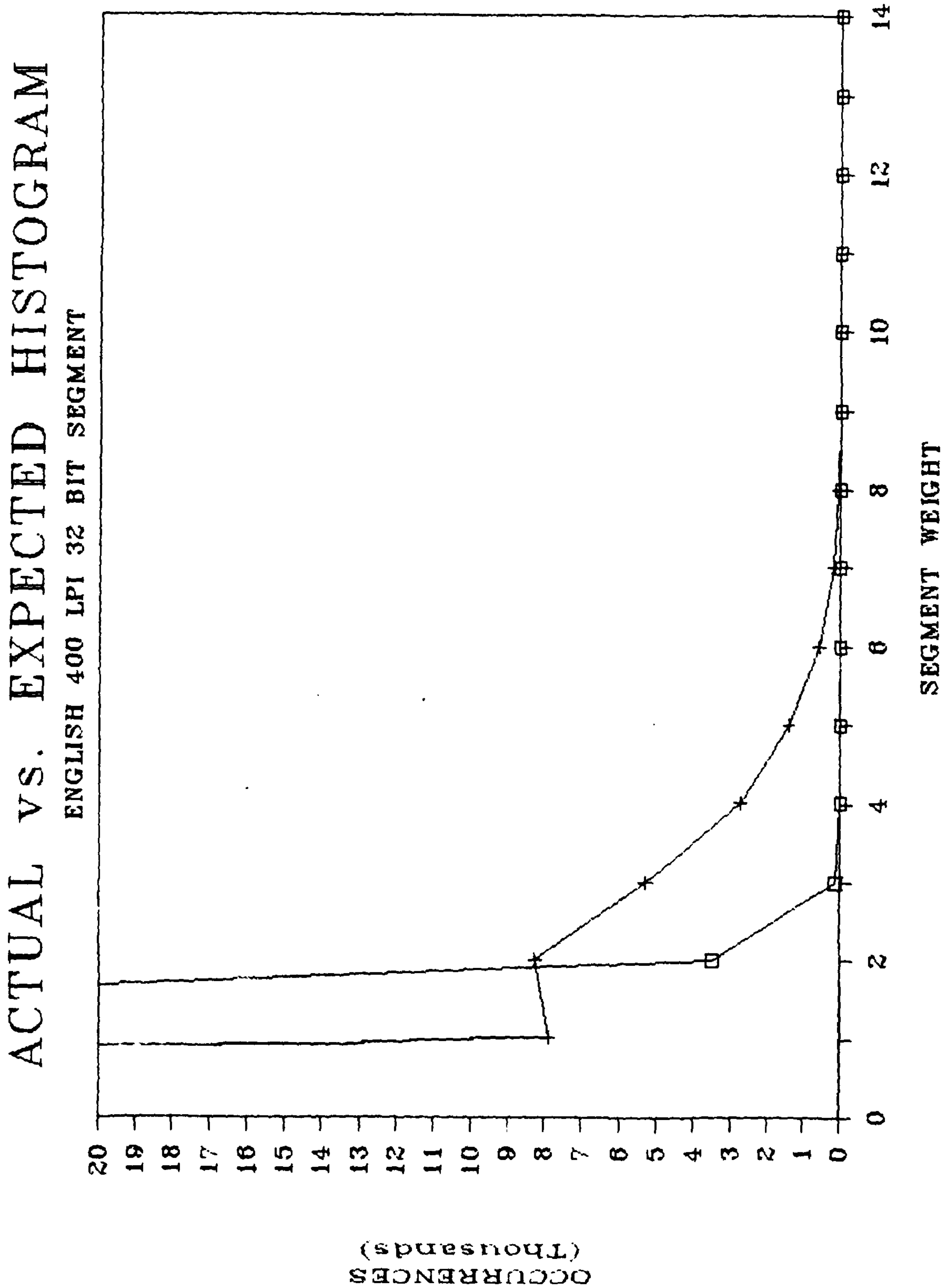


Figure 5.24 - Enlarged Section

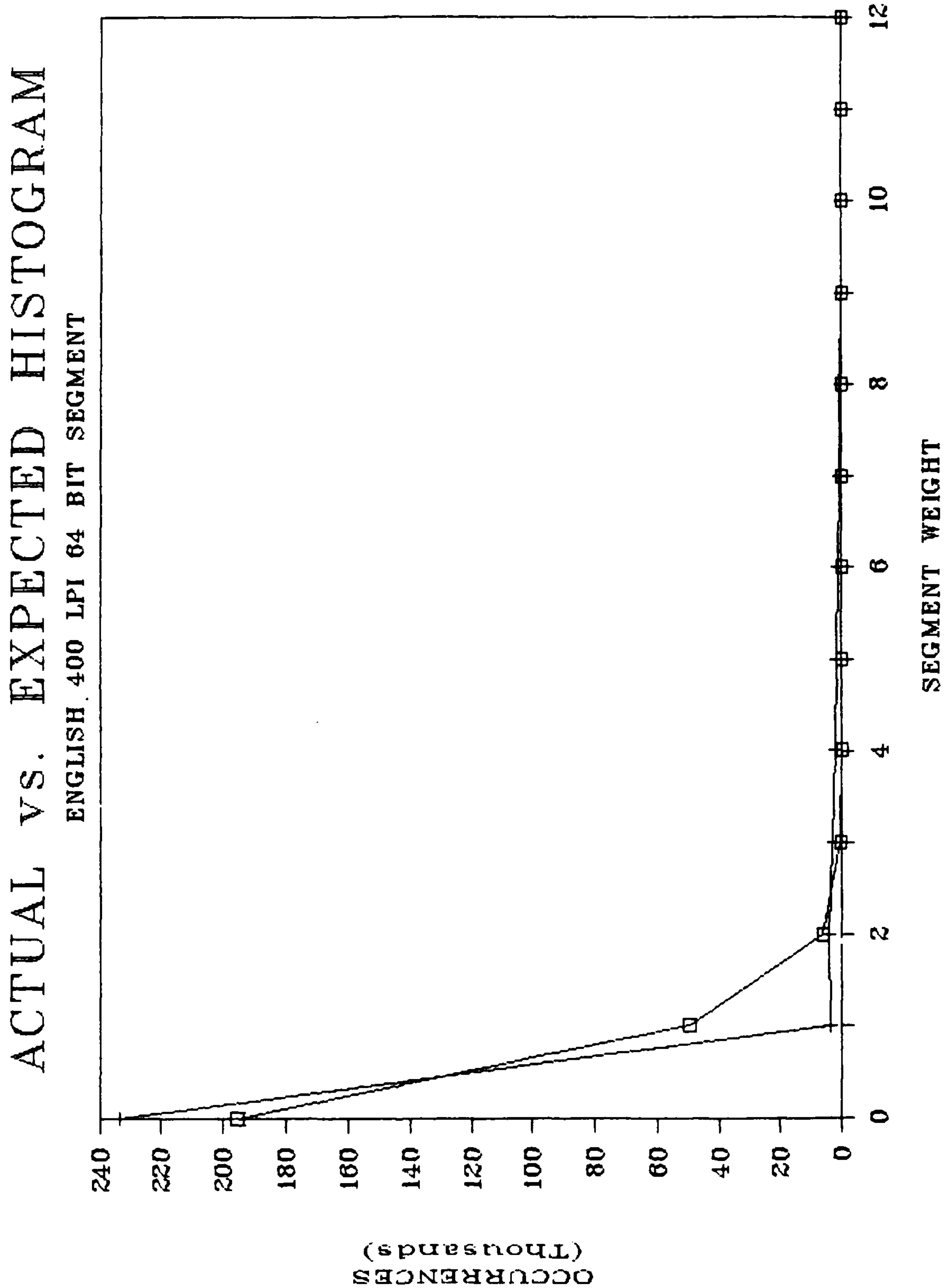


Figure 5.25 - Full Scale

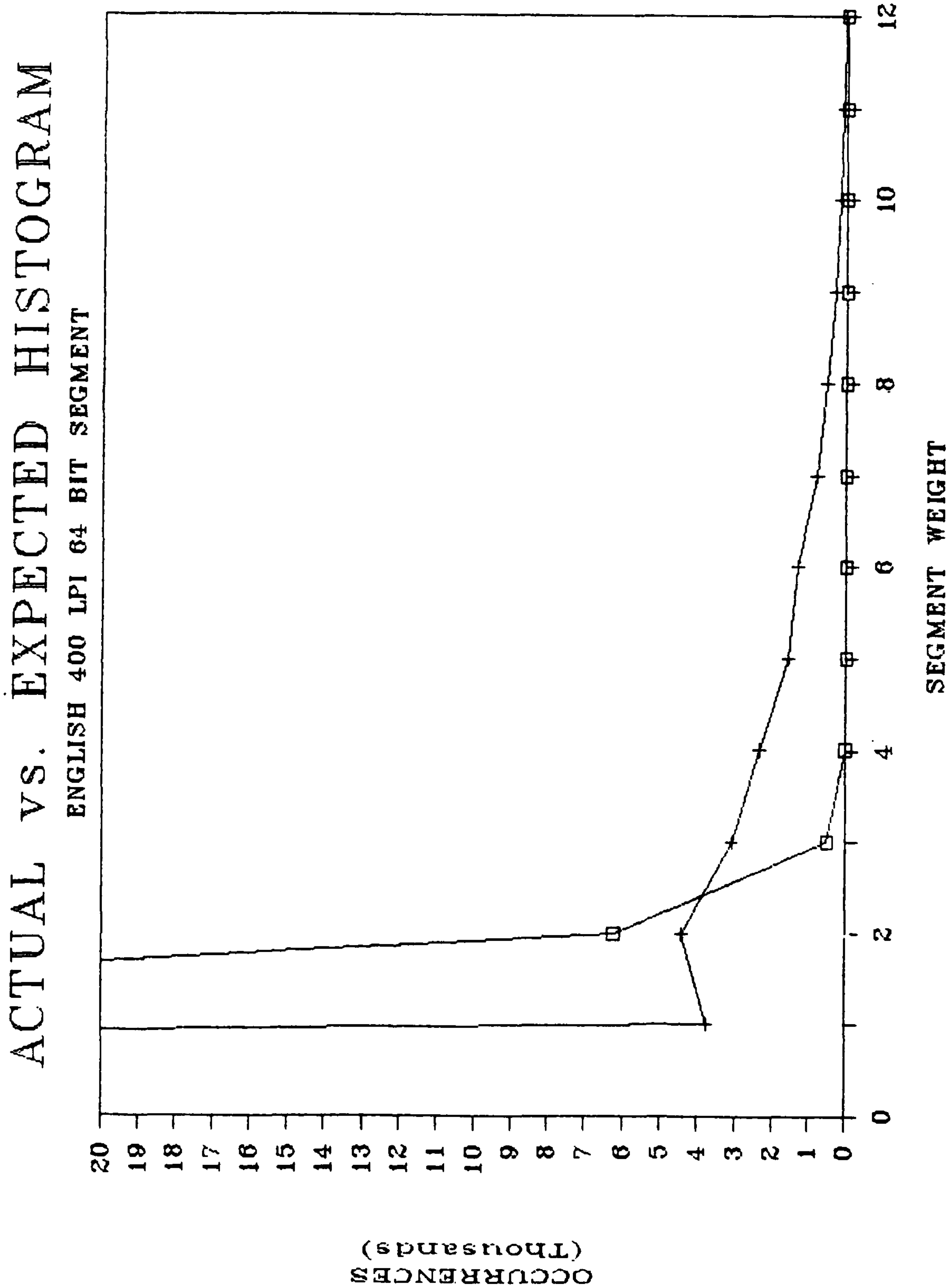


Figure 5.26 - Enlarged Section

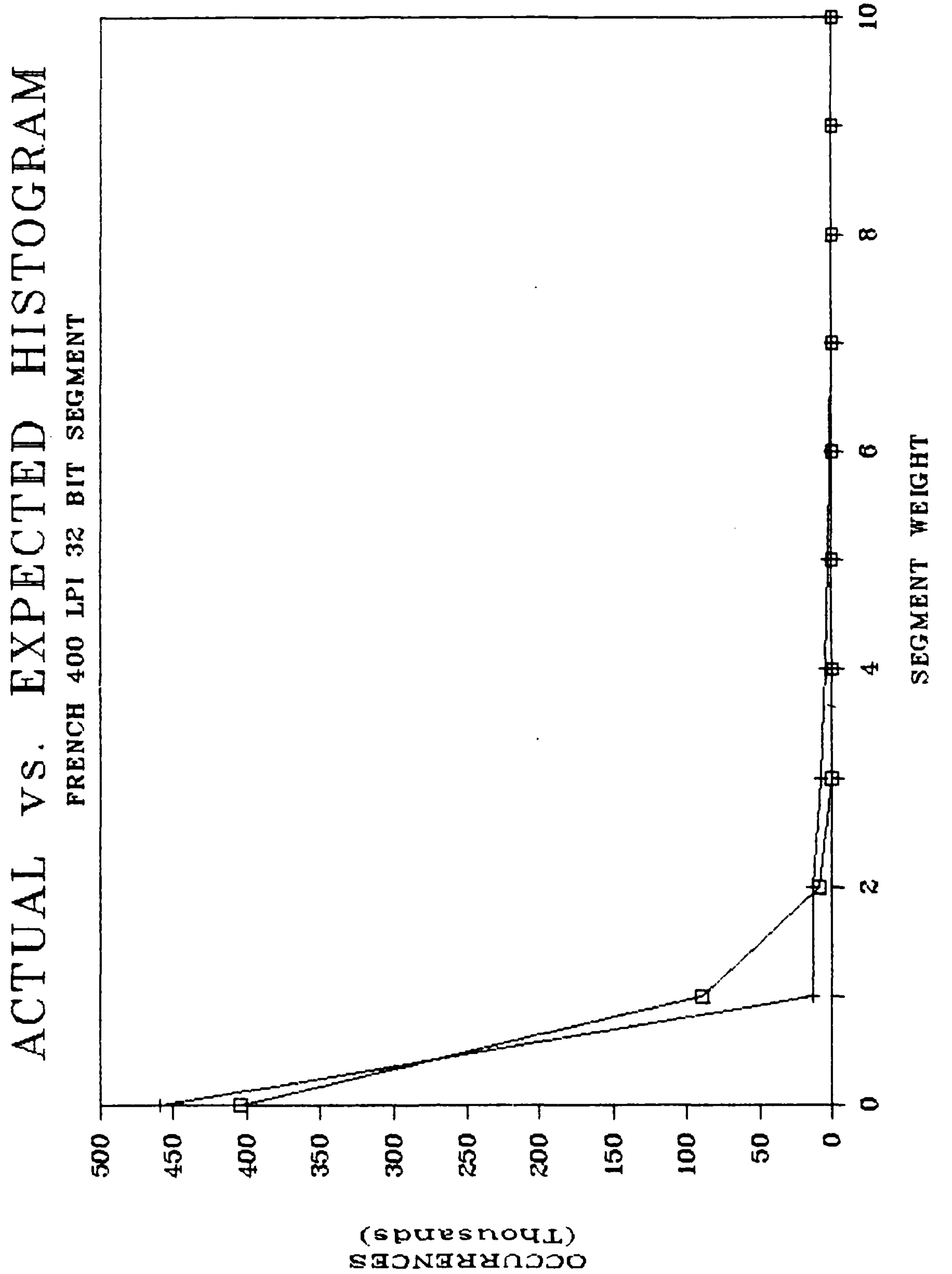


Figure 5.27 - Full Scale

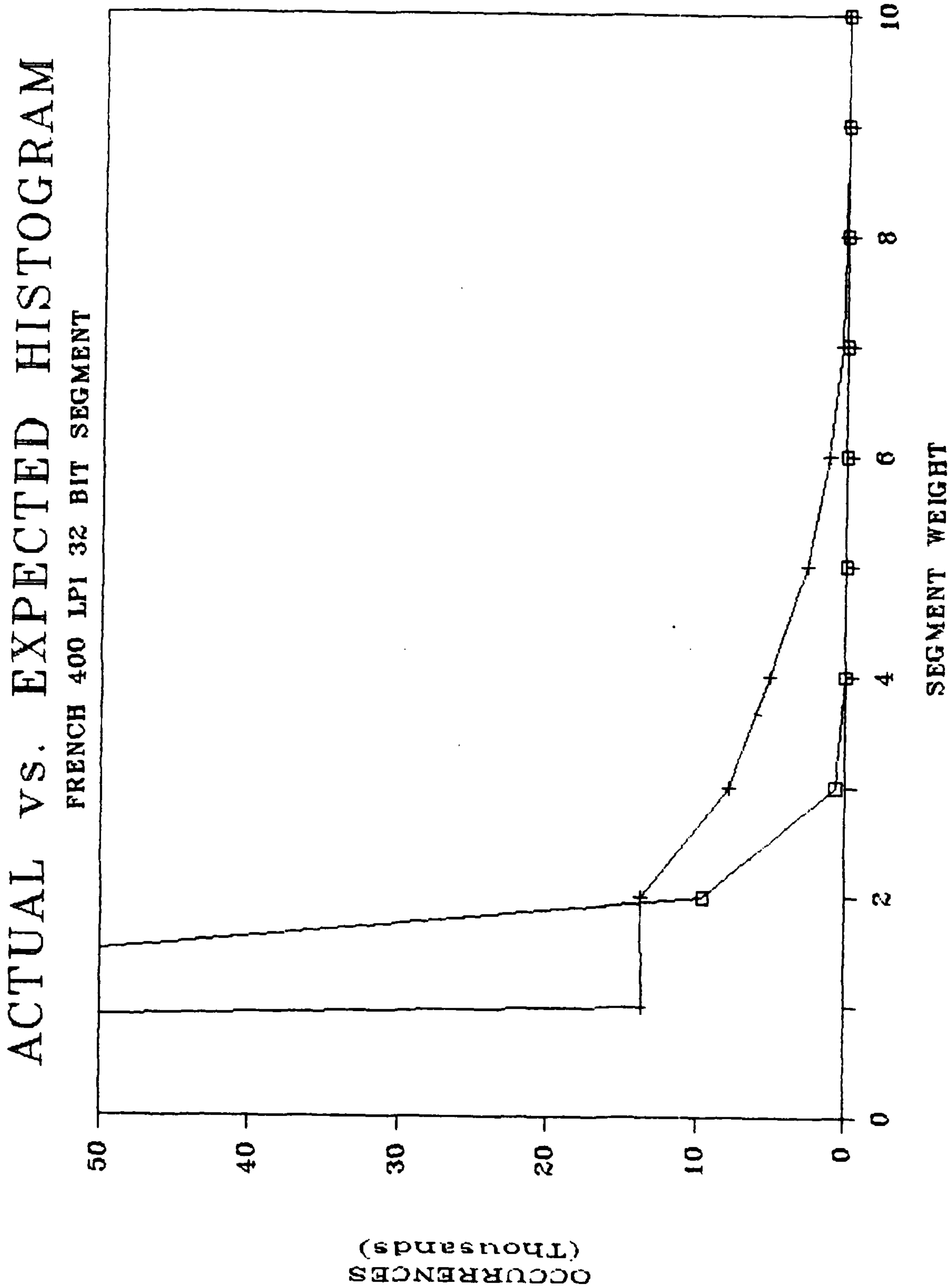


Figure 5.28 - Enlarged Section

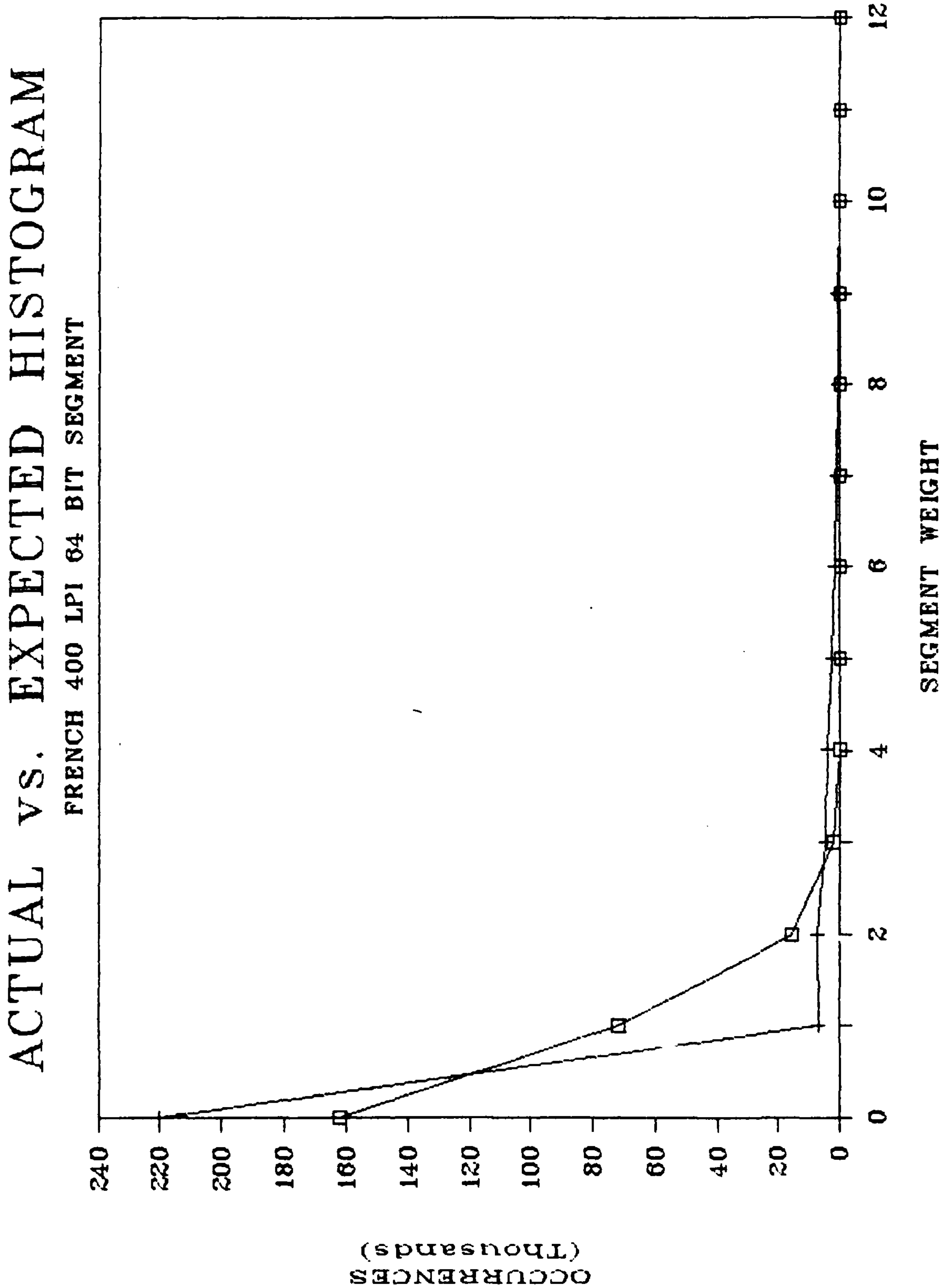


Figure 5.29 - Full Scale

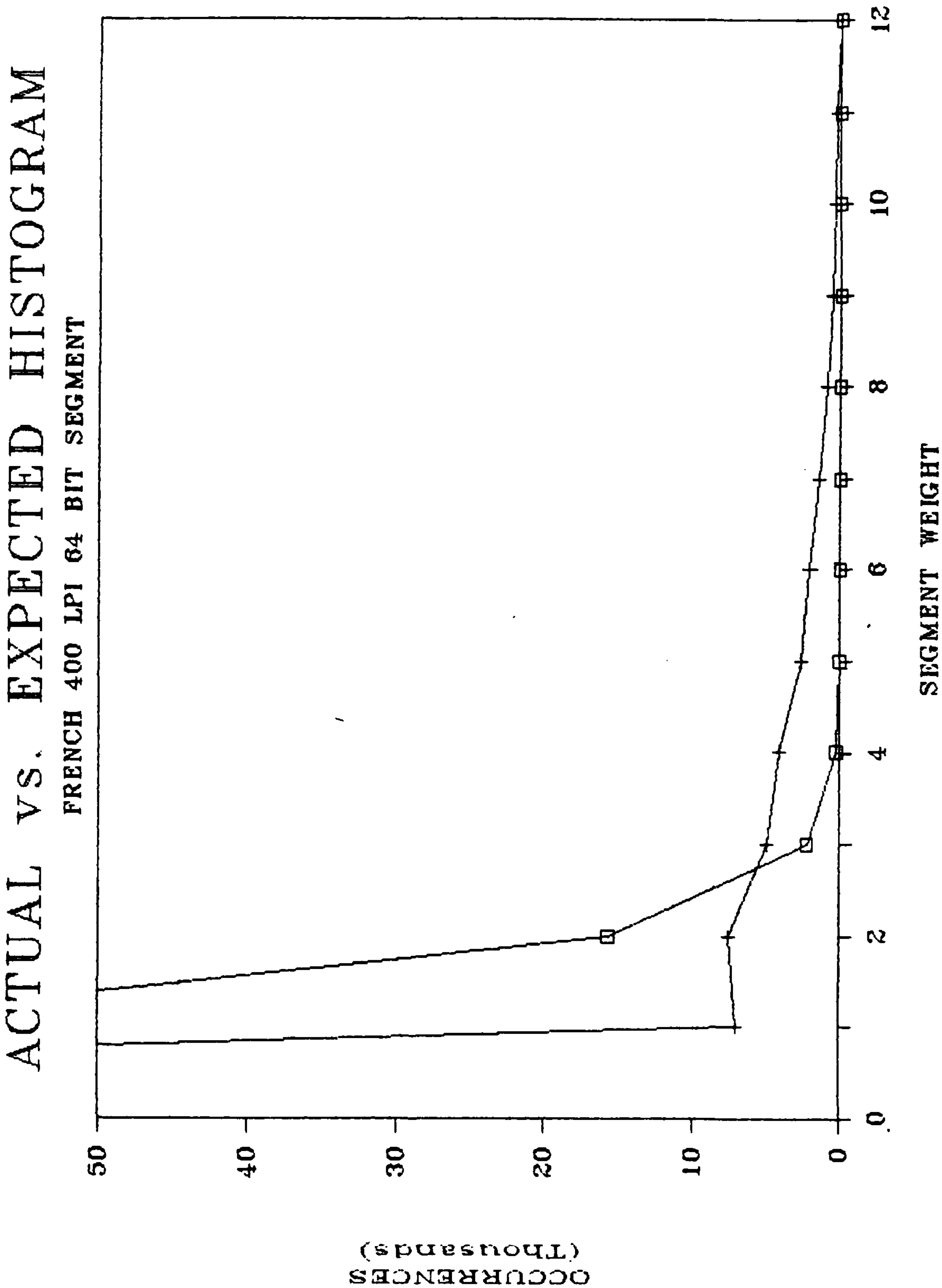


Figure 5.30 - Enlarged Section

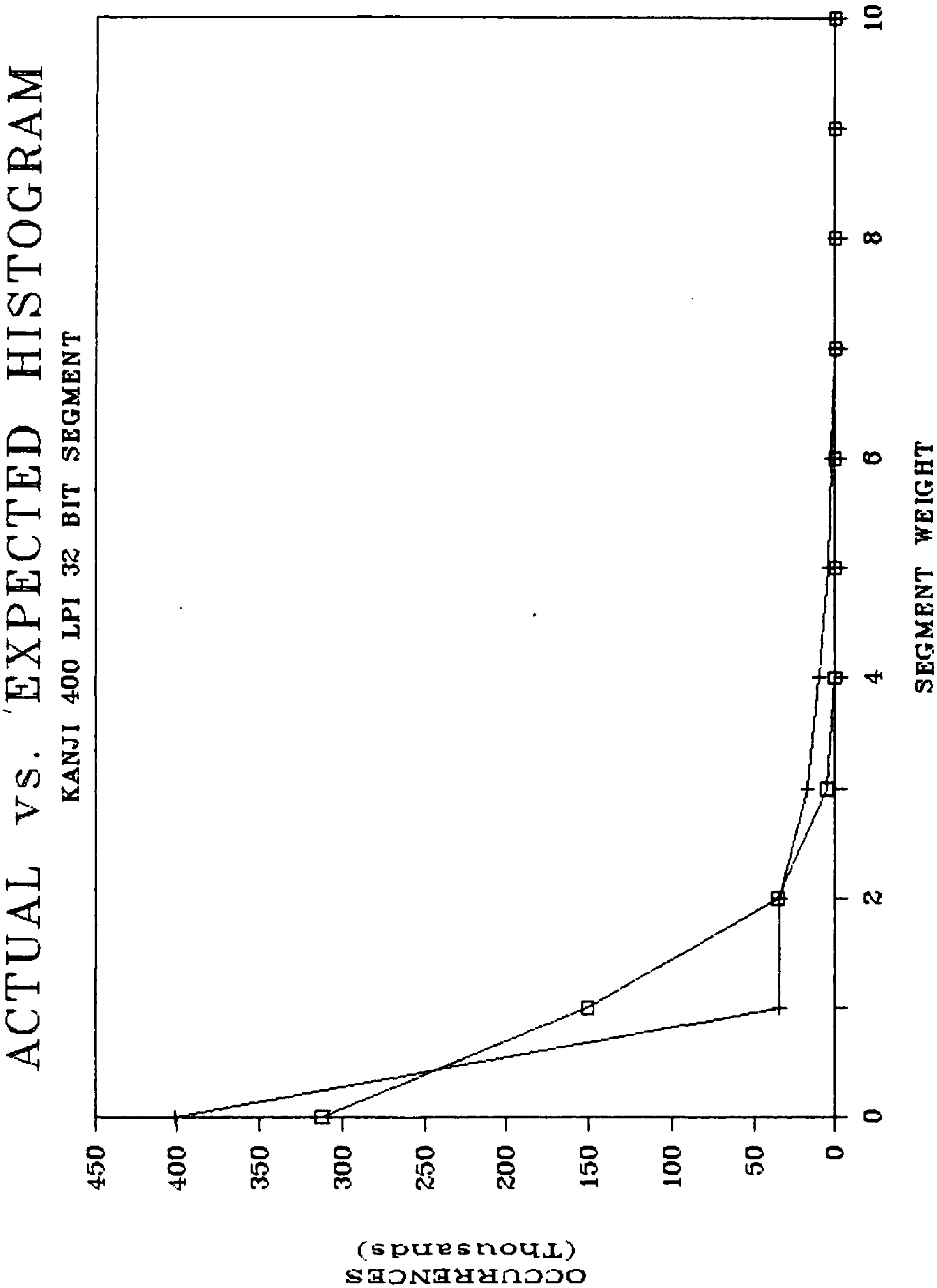


Figure 5.31 - Full Scale

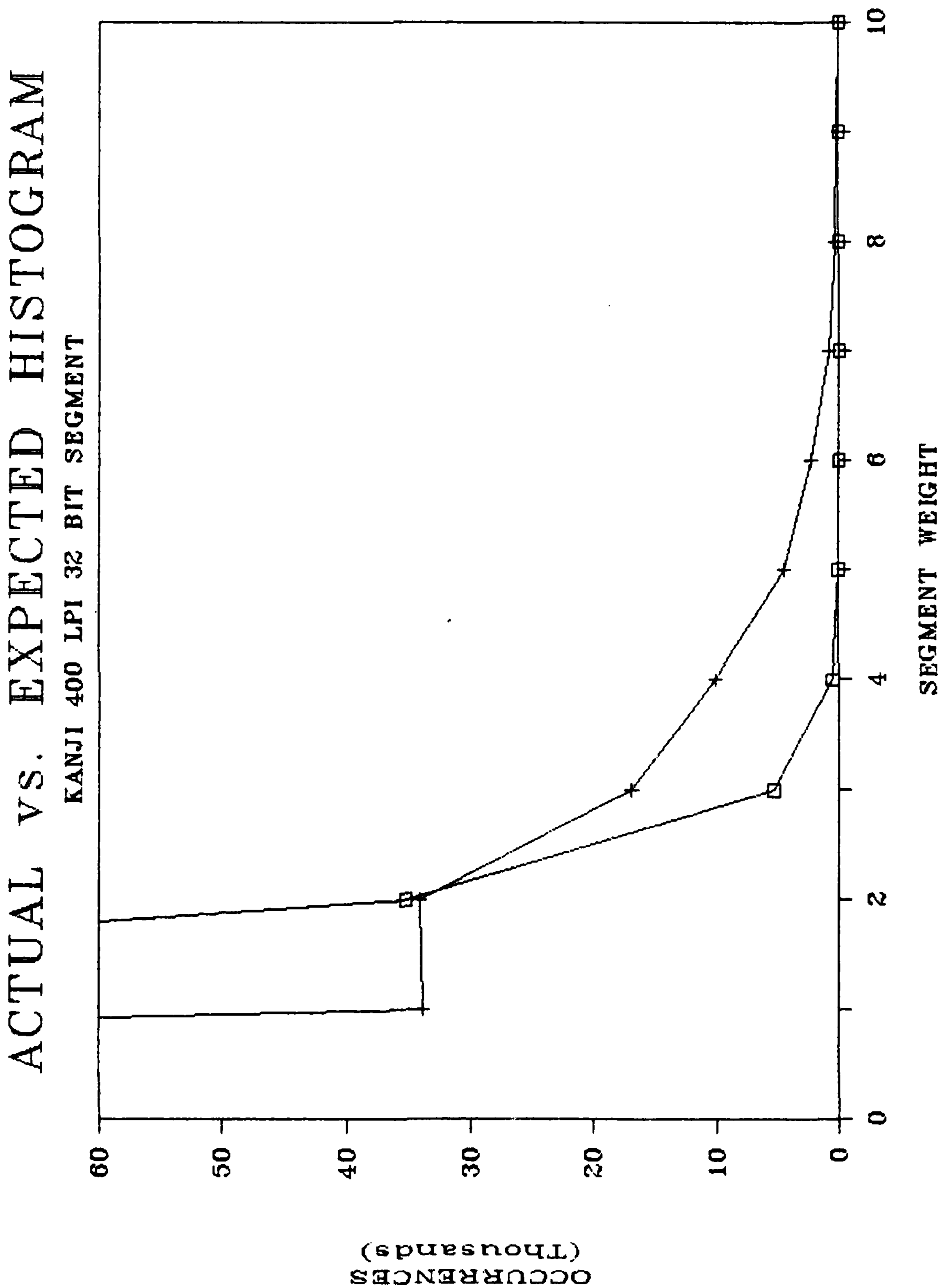


Figure 5.32 - Enlarged Section

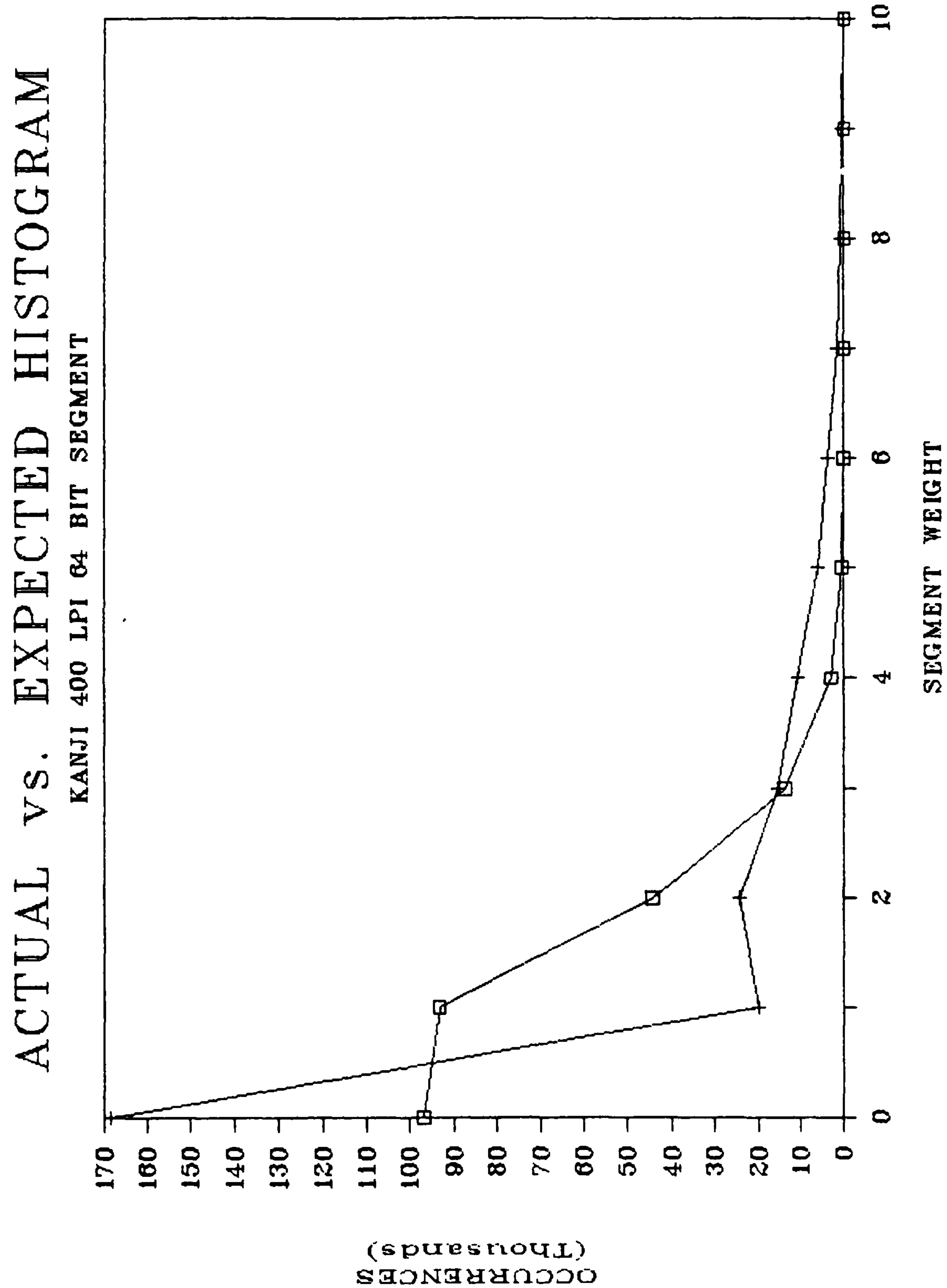


Figure 5.33 - Full Scale

6.0 Conclusions and Recommendations

In simulation runs of the SLDC algorithm, the overall compression achieved by compressing CIF's using the SLDC encoding algorithm did not compare favorably with the higher compression achieved by the MODREAD II algorithm run on the same original images. A side-by-side comparison of the results of the two images is presented in Table 6.1; the large difference in compression between the two algorithms is made more evident when displayed graphically (Figures 6.1 through 6.4; in the comparison graphs, the □ indicates the SLDC compression results and the + indicates the MODREAD II compression results.).

Compression differences between individual images show the Kanji documents to have the closest compression to MODREAD II, followed by the English, then French documents. The difference in compression between the two algorithms increases as the resolution increases. The compression vs. resolution curves of both algorithms are linear, with slope differences varying by a factor of two on the average, with MODREAD II having the steeper, more positive slope.

A possible suggestion to reach better compression, closer to MODREAD II, would be a better predicting function, as mentioned earlier, to better match the actual statistics of the CIF's. Another suggestion to consider is a new CIF prediction scheme which would generate CIF statistics that would more closely match those of a binomial distribution.

CCITT IMAGE	Resolution, LPI	SLDC Compression Ratio	MODREAD II Compression Ratio	% Diff
#1	200	20.17	30.57	34.0
English	240	23.16	36.54	36.6
Letter	300	27.43	45.44	39.6
	400	33.45	59.57	43.8
#5	200	11.61	17.61	34.1
French	240	12.90	21.00	38.6
Journal	300	15.10	25.91	41.7
	400	18.81	34.55	45.5
#7	200	5.44	7.59	28.3
KANJI	240	6.23	9.12	31.7
TEXT	300	7.34	11.43	35.8
	400	9.40	15.50	39.4
	200	9.39	13.56	30.8
AVERAGE	240	10.67	16.25	34.3
	300	12.56	20.26	38.0
	400	15.84	27.22	41.8

Table 6.1 - SLDC vs. MODREAD II Comparison

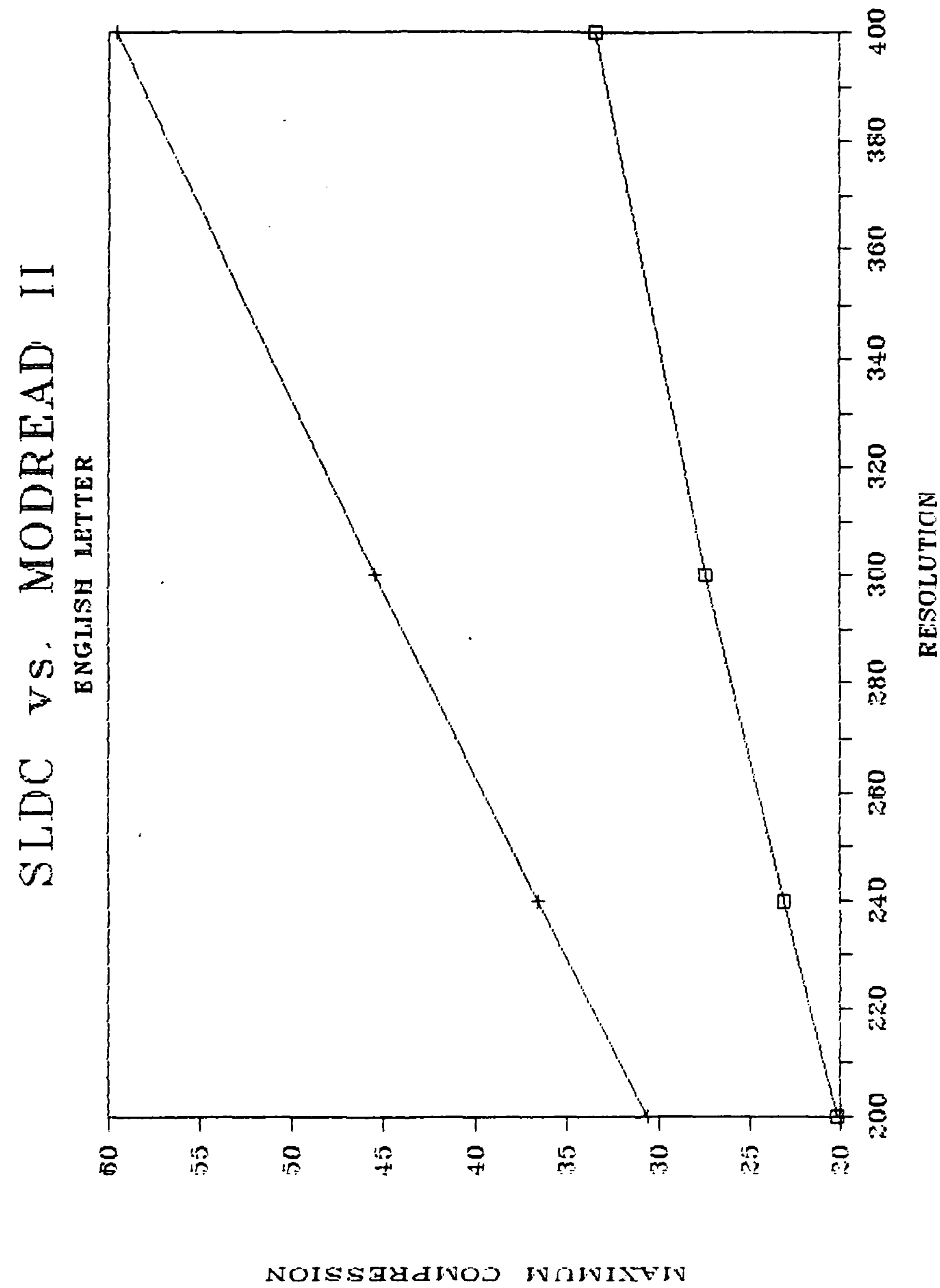

SLDC VS. MODREAD II
ENGLISH LETTER
RESOLUTION
MAXIMUM COMPRESSION
200 220 240 260 280 300 320 340 360 380 400
20 25 30 35 40 45 50 55 60

Figure 6.1

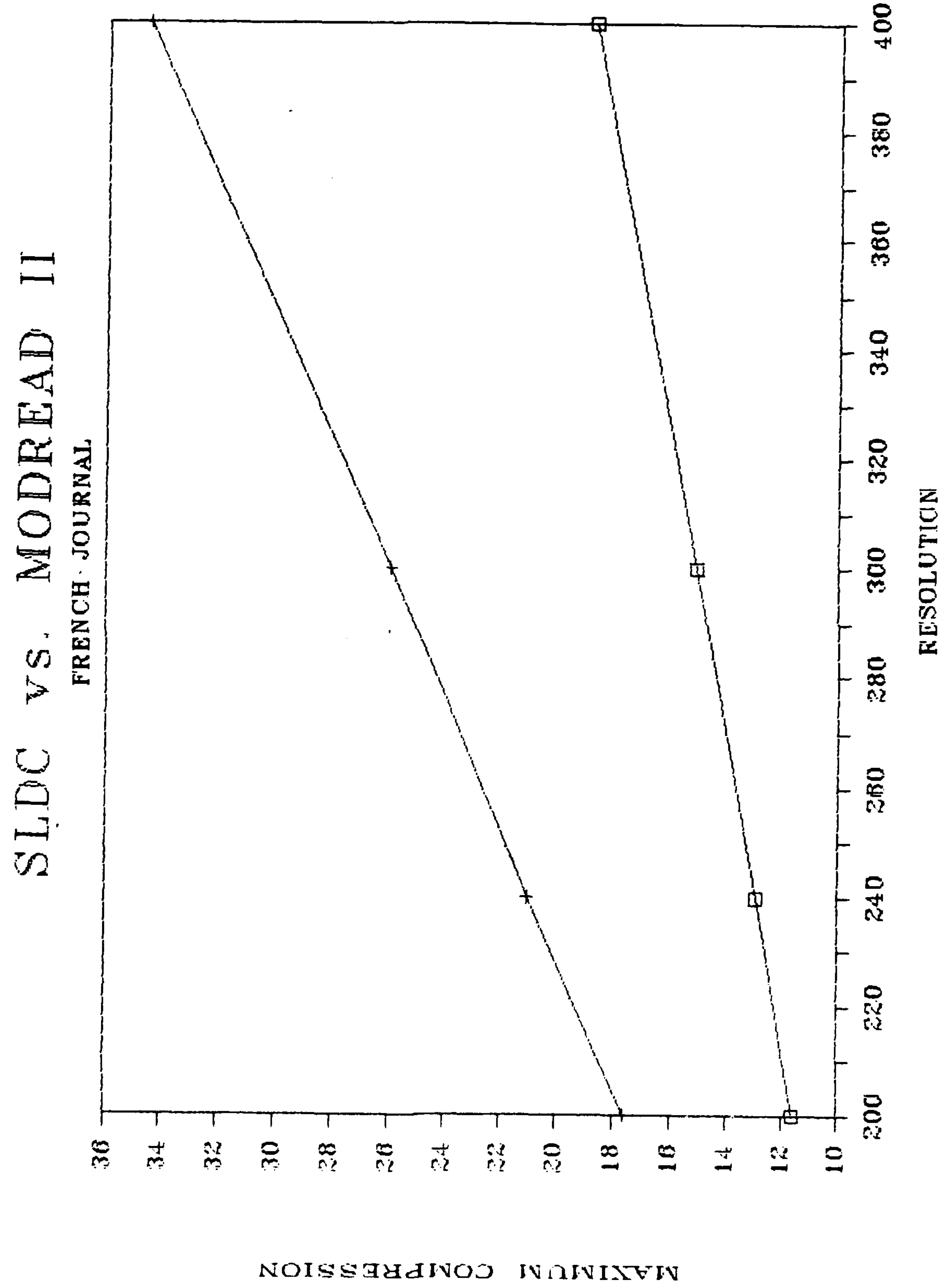
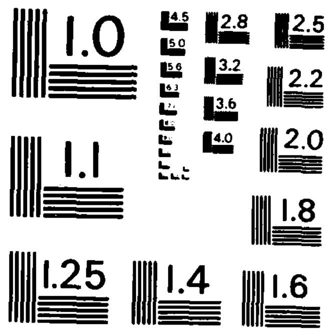

SLDC VS. MODREAD II
FRENCH JOURNAL
RESOLUTION
MAXIMUM COMPRESSION
200 220 240 260 280 300 320 340 360 380 400
10 12 14 16 18 20 22 24 26 28 30 32 34 36

Figure 6.2

AD-A160 846 SCAN LINE DIFFERENCE COMPRESSION ALGORITHM SIMULATION STUDY(U) DELTA INFORMATION SYSTEMS INC HORSHAM PA AUG 85 NCS-TIB-85-6 DCA100-83-C-0047 2/2

UNCLASSIFIED F/G 9/2 NL

END
FILMED
DTIC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

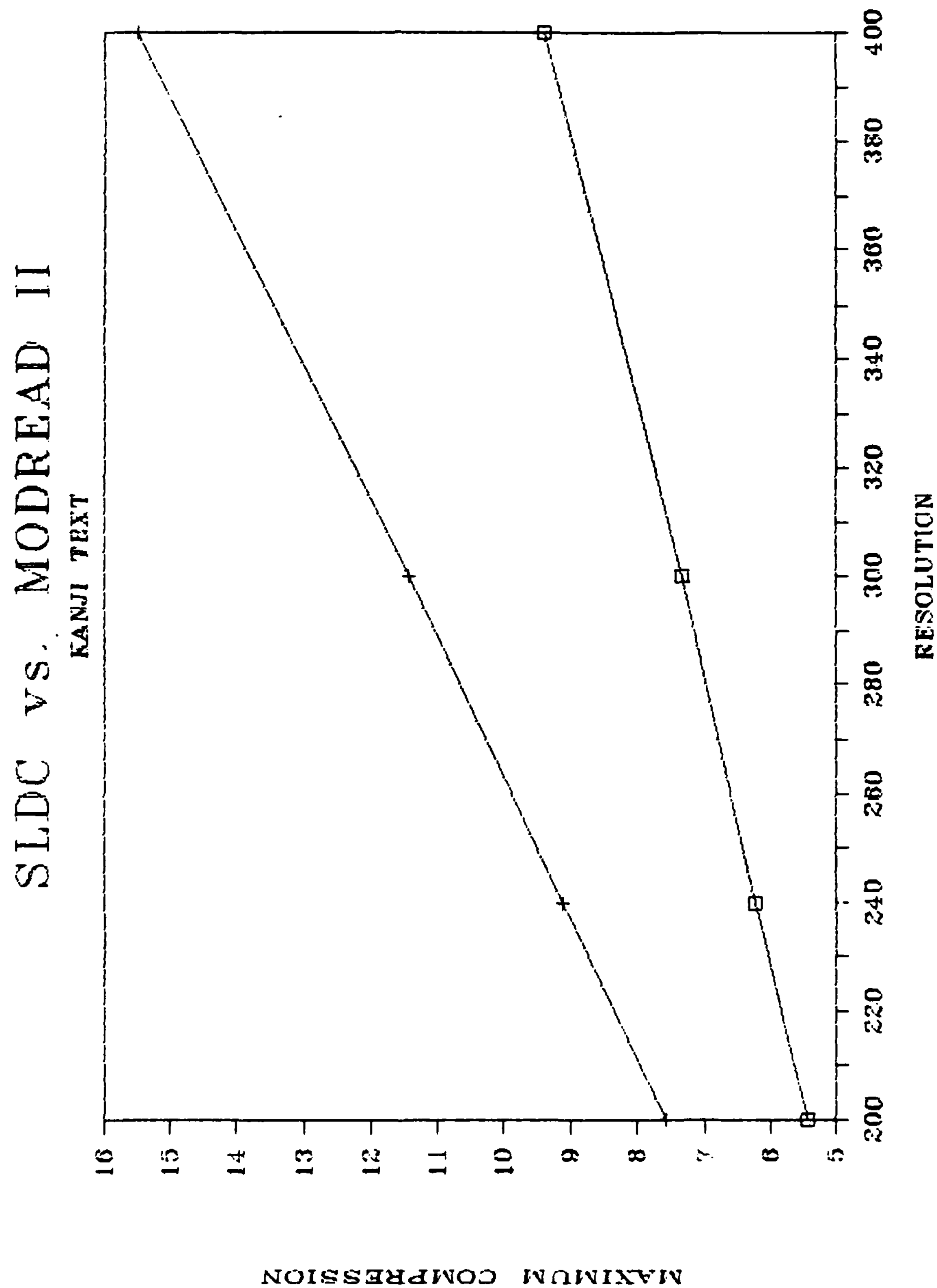
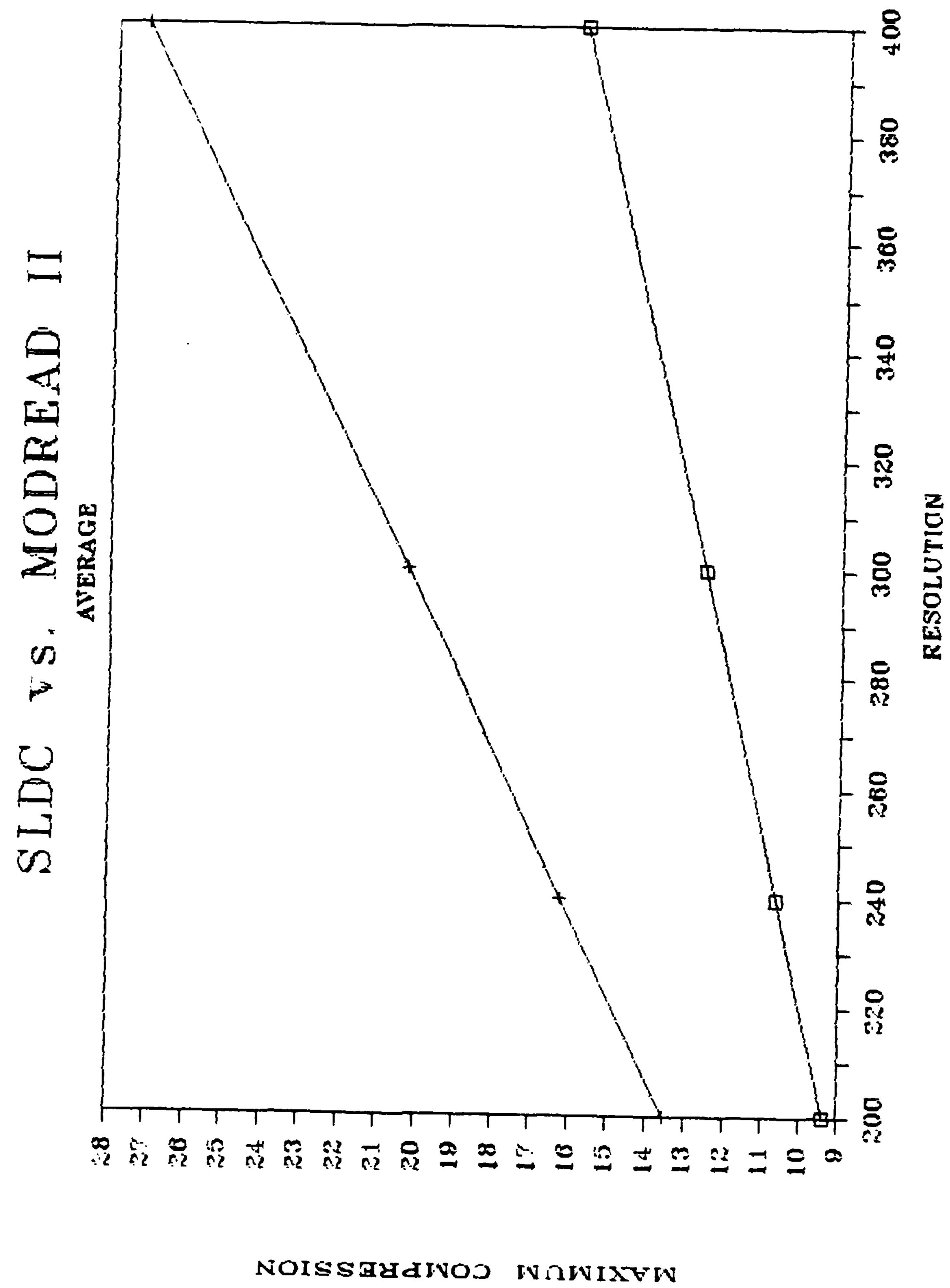

SLDC VS. MODREAD II
KANJI TEXT
MAXIMUM COMPRESSION
RESOLUTION
5
6
7
8
9
10
11
12
13
14
15
16
200
220
240
260
280
300
320
340
360
380
400

Figure 6.3


SLDC vs. MODREAD II
AVERAGE
RESOLUTION
MAXIMUM COMPRESSION
200
220
240
260
280
300
320
340
360
380
400
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

<u>Figure 6.4</u>

APPENDIX A

DESCRIPTION OF SLDC ALGORITHM

APPENDIX A -- DESCRIPTION OF SCAN LINE DIFFERENCE COMPRESSION ALGORITHM

A-1. INTRODUCTION

This Appendix describes a Scan Line Difference Compression (SLDC) algorithm which is hypothesized to be an efficient distortionless data compression algorithm as well as a technique which can easily be implemented using existing microcomputer and LSI technology. An experimental test of this technique is about to be initiated. Until these experimental results are available, it shall be necessary to rely on "arm-waving" arguments to explain why the SLDC algorithm is considered to be a viable candidate for facsimile data compression. These arguments are given in Section A-3.

A-2. DESCRIPTION OF SLDC ALGORITHM

Figure A-1. is a diagram of the overall image transmission system. As shown on this figure, the overall system is partitioned into:

1. the source coding subsystem, concerned with the minimization of the number of bits necessary to permit the distortionless reconstruction of an image, and,

2. the channel coding subsystem, which attempt to correct whatever errors may be introduced during the signal transmission process.

```
                 -----------SLDC Encoder---------

|  Image  |      |Conditioned|     |Conditioned|     | Error Control |
| Source  |------| Scan Line |-----| Scan Line |-----|   Encoder &   |
|_________|  |   | Generator |  |  |  Encoder  |  |  |  Transmitter  |
             |                  |                 |
             |                  |             Compressed
        Sequential         Conditioned        Conditioned     + --Noise
        Scan Lines         Scan Lines         Scan Lines
             |                  |                 |
|  Image  |  |   | Scan Line |  |  |Conditioned|  |  | Receiver and  |
| Sink    |------|   Recon-  |-----| Scan Line |-----| Error Control |
|_________|      | struction |     |  Decoder  |     |    Decoder    |

                 -----------SLDC Decoder---------
```

Figure A-1. -- Overall Data Compression Process

This Appendix deals exclusively with the source coding aspects and presumes that the transmission process is noiseless either due to the inherent absense of noise or an effective channel coding subsystem providing the necessary error control.

The SLDC system consists of an encoder, located at the image generation source, and a decoder, located at the point where the image is regenerated. The encoder and decoder represents a transform pair in which the decoder performs the inverse process to that performed by the encoder. For most of the remaining discussion, the SLDC algorithm shall be described in terms of the encoding functions only, since that effectively defines the entire process.

It is assumed that the image to be compressed can be represented as a two dimensional rectangular binary matrix composed of "r" rows and "c" columns. Every matrix element is either white (0) or black (1). The raw image can therefore be expressed with a binary sequence of rc bits in length. The image is output from the source as r successive scan lines, where each scan line consists of a string of c bits.

If the process generating the image were memoryless (i.e., future image elements were independent of all past image elements) and if the probability of a "1" or a "0" at each element were 0.5, then distortionless compression would not be effective and no better scheme than the transmission of the raw rc image bits would be possible. However, in typical images to be transmitted, there is considerable inherent redundancy in the raw image. In these cases, it is generally possible to transmit an encoded replica of the image with T bits (T less than rc) such that the image may be exactly reconstructed using only the T transmitted bits. The efficiency of the compression process can be measured by the compression ratio, R_C, given by

$$R_C = rc/T \qquad \text{[EQ. A-1]}$$

The SLDC algorithm for the distortionless compression of an image consists of a Conditioned Scan Line Computation step and a Compression step.

A-2.1 Conditioned Scan Line Computation Step.

The purpose of this step is to process an image consisting of a number of scan lines and to generate a Conditioned Image File (CIF), composed of a set of Conditioned Scan Lines (CSLs), subject to the following conditions:

1. the original image can be reconstructed from the CIF without distortion, and,

2. the information theoretic entropy of each CSL is approximately minimized subject to practical processing limitations.

Although these two conditions may appear to be contradictory, they really are not. The first condition requires that the CIF, consisting of the set of CSLs, must be transformable back to the set of original scan lines without distortion. The second condition is concerned only with the compressability of the image considered one scan line at a time. The conditioning procedure should transform the original scan line into a CSL having lower entropy than the original when considered on an individual scan line basis.

The conditioning algorithm shall generate a prediction of a scan line element based upon the state of the four neighboring and previously scanned image elements as shown in Figure A-2.

```
                  Direction of scan--
Previous Scan Line:    . . . A B C . . .
 Current Scan Line:    . . . D X   . . .
                             |
                             |—Element to be predicted
```

Figure A-2 -- Nearest Neighbor Elements

The CIFs shall be generated based upon a State Machine specification that shall predict the state of each scan element based upon Bits A, B, C, and D where "0" and "1" represent the "expected" and "unexpected" state respectively. A ring of "0" bits shall be presumed to encircle the perimeter of the raw image. This is relevant only when the element being predicted is on the image boundary. The prediction condition algorithm shall be completely defined by a state table that has not yet been determined.

A-2.2 Compression Encoding Step.

The second step of the SLDC encoding process represents the actual compression process. For each Conditioned Scan Line (CSL) string to be compressed, the following substeps shall occur:

1. Encode the weight (W) of CSL. If W=0, proceed to next CSL; otherwise, proceed to Substep 2.) below.

2. Segment CSL into m segments of n-bits each (mn=c) where n is a design parameter [NOTE: if c is prime or not divisible by a convenient integer, it may be necessary to pad CSL by appending some "0"-bits in order to achieve a convenient divisor], and,

3. Compute and encode for transmisssion a Segment State Word (SSW) for each of the m segments. Each of the SSW shall uniquely define the state of the respective segment and shall, in general, be encoded into fewer than n-bits. A SSW contains the following two variable length fields:

    a. WEIGHT, the weight of the segment, and,

    b. RANK, an index value which uniquely specifies the bit positions of all "1"-bits within segment. If the WEIGHT-value = 0, the RANK-field is omitted.

After the last non-null segment of a CSL has been encoded, the remaining SSWs may be omitted. This can be recognized by maintaining a running count of processed "1"-bits within the CSL and comparing this to W. The format of an Encoded Scan Line (ESL) is shown in Figure A-3.

Scan Line Weight	SSW(1)	SSW(2)	. . .	SSW(J)	. . .	SSW(n)

WEIGHT(J)	RANK(J)

Figure A-3. -- Encoded Scan Line Format

The key elements of the SLDC encoding consists of the efficient manner in which the WEIGHT(·) and RANK(·) are encoded. This is described in the following two sections.

A-2.2.1 Segment Weight Encoding Procedure.

Let p be the average probability of a "1"-bit within a CSL. By definition,

$$p = W/mn \qquad \text{[EQ. A-2]}$$

The segment weight shall be encoded using a variable length Huffman code. This type of code is extensively described in most information theory texts (e.g., Ref. 2). If P_k is defined to be the probability that the weight of an n-bit segment is k, then Huffman coding shall minimize the average length of the segment weight field. Thus,

$$\sum_{k=0}^{n} P_k L_k = \text{minimum} \qquad \text{[EQ. A-3]}$$

where L_k = number of bits in WEIGHT-field specifying a segment of weight k.

For reasons to be explained in Section A.2.2.2, the maximum weight segment that shall be handled by the normal SLDC process is k' where k' is listed in Table A-1. Segments of weight greater than k' shall be handled by special procedures to be described later.

It shall be assumed that the W "1"-bits are binomially distributed within CSL. Hence,

$$P_k = \text{Prob[segment weight} = k] = C(n,k)p^k(1-p)^{n-k} \qquad \text{[EQ. A-4]}$$

where $C(n,k) = \dfrac{n!}{k!(n-k)!}$

The binary Huffman code is computed for the set of events with respective probabilities,

$$P_0, P_1, \ldots, P_{k'}, \text{ and } S_{k'}$$

where

$$S_{k'} = 1 - \sum_{k=0}^{k'} P_{k'}. \qquad \text{[EQ. A-5]}$$

After the codebook has been computed, the codewords shall be used to define the segment lengths for the current CSL. Note that p shall typically vary from one scan line to the next and therefore either a set of codebooks must be prestored corresponding to all possible values of p or the codebook must be recomputed each scan line.

For segments whose weight (k) is less than or equal to k', the segment weight is identified by use of the codeword corresponding to probability P_k. If the segment weight exceeds k', then the codeword for probability $S_{k'}$ shall be encoded followed by the n-bit segment in uncompressed form. In this case the rank encoding process, described in the following section is omitted.

A-2.2.2 <u>Rank Encoding Procedure</u>.

The <u>rank</u> of an n-bit segment of weight k shall be defined to be the number of possible n-bit sequences of weight k that are numerically less than the segment being encoded. The rank encoding process can be implemented recursively by successively scanning the bits of the segment from most significant bit to least significant bit and, whenever a "1"-bit is encountered, to accumulate a count of the number of patterns which are already known to be numerically inferior to that of the segment. The rank field is omitted if k=0 and, if k is greater than 0, the rank is encoded into a field of length $[\log_2 C(n,k)]$-bits where [X] represents the smallest integer greater than or equal to X.

The following example may help explain the procedure. Let the segment length and weight be n and k respectively (k greater than 0). The process is initialized by clearing a counter, named RANK, to zero. Then, starting at the most significant bit, the process recursively scans the segment from left to right as shown in Figure A-4, and increments RANK by an appropriate constant whenever a "1"-bit is encountered. For example, suppose the scanning process has progressed to the point where Bit-I is about to be examined and that j "1"-bits (0 j k) have previously been encountered in the scanning process from Bit-(n-1) to Bit-(I+1) inclusively. There are therefore (k-j) "1"-bits within the segment that have not yet been discovered. Bit-I is now examined and, if Bit-I = 0, the scan process immediately moves to the next bit, Bit-(I-1). If Bit-I = 1, then the current segment is known to be numerically superior to all C(I,k-j) sequences which have their low order (k-j) "1"-bits confined to the I low order bit positions (i.e., Bit-0 to Bit I-1 inclusive). Therefore RANK shall be incremented by C(I,k-j) and j shall be incremented by 1 (in that order) before proceeding to the next scan position.

The rank encoding process terminates when either all k "1"-bits have been found by the scanning procedure OR there are exactly j "1"-bits that have not yet been found AND there are only j bit positions remaining to be scanned. In the latter case, all unscanned bit positions must contain "1" and therefore have the lowest possible rank. Hence, since the rank increment would be zero, the remaining scanning steps can be bypassed. Annex-1 contains a numerical example of the rank encoding process.

```
------------ n-bit segment of weight k -----------

Bit Number |n-1|n-2|    . . .   | I |              | 0 |
             |                    |                  |
            Most             Current bit           Least
         Significant        being examined      Significant
            Bit                                     Bit
             Direction of bit scan ---
```

Figure A-4. -- Rank Encoding Scanning Process

It shall be presumed that the encoding algorithm is capable of performing arithmetic integer computation with a maximum word length of either 16 or 32 bits. If the segment length is greater than the arithmetic word length, the RANK-value might cause an arithmetic overflow. For this reason, the value of k' shall be limited to the following maximum values to avoid the possibility of an overflow condition.

Segment length in bits	Maximum value of k'	
	16-bit word length	32-bit wordlength
18 or less	unconstrained	unconstrained
20	6	"
24	5	"
28	4	"
32	4	"
36	4	14
40	3	11
44	3	10
48	3	9
52	3	8
56	3	8
60	3	8
64	3	7

Table A-1. - Maximum Weight Segment for Normal SLDC Processing

A-3 RATIONALE FOR SLDC ALGORITHM

The effectiveness of a facsimile data compression system must necessarily be based upon the performance/cost tradeoff with respect to documents representing a realistic traffic load. It is the purpose of the present investigation to subject the SLDC algorithm to such a test using document images believed to be representative of commercial requirements. Until the results of this investigation are completed, the effectiveness of the SLDC algorithm remains in doubt.

Since the forthcoming investigation shall require money and other resources, it is reasonable to insist upon some rationale for the potential benefits that may be gained before embarking on this investigation. There are an infinite number of possible compression algorithms. Why should the Government expend resources to investigate the SLDC technique? While an insistence on a "proof-of-effectiveness" as a prerequite for the investigation is impossible and would represent a Catch-22 dilemma, the following is offered as a plausibility argument in defense of the SLDC algorithm.

The SLDC represents a two step process. In the first step the image is reduced to a set of conditioned scan lines (CSLs), each of which are constructed so that they contain only the "unexpected" changes in image pattern relative to the previously scanned neighboring image elements. A pattern extension that can be implied from the previously scanned image need not be transmitted since the decoder could reconstruct the image extension without guidance from the encoder. It is only when the decoder would incorrectly construct the image that guidance is needed from the encoder to override the decoder's default algorithm. Unexpected changes are signalled by a binary "1"; the absense of an unexpected change is encoded as a "0". It is expected that the investigation performed under the Section 3.1 subtask shall develop an efficient procedure that minimizes the frequency of unexpected changes. It seems reasonable that a conditioning technique which predicts the binary state of an image element based upon some weighted average of four of the element's nearest neighbors should be correct most of the time.

The second step of the SLDC algorithm represents the actual compression process. Each CSL to be compressed is fragmented into a series of n-bit segments. For each segment a variable length Huffman code is used to encode the weight (k) of the segment. Huffman codes are known to be optimal in the sense that no other distortionless code is possible which can encode the data into a lesser average number of bits. Assuming that each of the possible C(n,k) weight-k patterns of length n-bits are equally likely, the rank encoding procedures is also theoretically optimal in encoding the position of the k "1"-bits in the segment. Thus, each of the components of the encoding process is optimal in some sense; however, this does not imply that the entire algorithm is globally optimal. Indeed it is intuitively obvious that some non-optimality is introduced at each stage of modularization. The SLDC algorithm has performed the following modularization steps:

1. the fragmentation of an image into CSLs,

2. the fragmentation of a CSL into segments, and,

3. the separate encoding of segment weight and specific segment pattern given a known weight.

While each of these partitioning operations shall undoubtedly reduce the compression ratio performance relative to a globally optimal compressor, the modularization is expected to reduce the computational complexity and may therefore allow the SLDC algorithm to approach the globally optimal compression scheme more closely than previously implemented compression scheme. Hence, although the performance and feasibility of implementation cannot be proven at this point, the SLDC procedure offers considerable promise and is considered worthy of a empirical demonstration which shall evaluate its effectiveness relative to other compression schemes.

ANNEX-1 -- NUMERICAL EXAMPLE OF RANK ENCODING PROCESS

In this Annex the rank encoding of a 9-bit segment of weight 3 shall be evaluated. In this case, n=9 and k=3. Suppose the segment pattern were 010100100 and let us establish the bit numbering convention as shown below.

Bit Number:	8	7	6	5	4	3	2	1	0
Bit State:	0	1	0	1	0	0	1	0	0

Initialize by setting RANK=0, j=0 (Number of previously encountered "1" bits), and the bit scan position to Bit-8.

Scan Position (I)	Bit State	No. of Found "1"-Bits (j)	C(I,k-j)	Rank Encoding computation
		0		RANK = 0 (INITIALIZATION)
8	0	0	56	RANK = 0
7	1	0	35	RANK = 0 + 35 = 35
6	0	1	15	RANK = 35
5	1	1	10	RANK = 35 + 10 = 45
4	0	2	4	RANK = 45
3	0	2	3	RANK = 45
2	1	2	2	RANK = 45 + 2 = 47
1	0	3 (j = k: TERMINATE with RANK = 47)		

The encoded length of the RANK field is $[\log_2 C(9,3)]=[\log_2 84]=[6.394]=7$. Thus the RANK-field is encoded as 0101111.

To validate this example, the table below lists all possible n=9, k=3 patterns in numerically increasing order up to and including the test case (010100100).

RANK	PATTERN	RANK	PATTERN	RANK	PATTERN	
0	000000111	16	000110001	32	001100100	
1	000001011	17	000110010	33	001101000	
2	000001101	18	000110100	34	001110000	
3	000001110	19	000111000	35	010000011	
4	000010011	20	001000011	36	010000101	
5	000010101	21	001000101	37	010000110	
6	000010110	22	001000110	38	010001001	
7	000011001	23	001001001	39	010001010	
8	000011010	24	001001010	40	010001100	
9	000011100	25	001001100	41	010010001	
10	000100011	26	001010001	42	010010010	
11	000100101	27	001010010	43	010010100	
12	000100110	28	001010100	44	010011000	
13	000101001	29	001011000	45	010100001	
14	000101010	30	001100001	46	010100010	
15	000101100	31	001100010	47	010100100	-- Test Case Segment

APPENDIX B

SOFTWARE OPERATING INSTRUCTIONS

PROGRAM: CHART

DESCRIPTION:

This program reads an image input file one line at a time recording occurrences of each nearest neighbor state vector and state of each bit for every bit in the file. The results are sent to the line printer in the form of Table 1 in Appendix A. Also an occurrence-black-white table of the same nature is calculated.

CALLING SEQUENCE

CHART,<INPUT NAME>, <PROBABILITY FILE NAME>

INPUT NAME - Input Image File

OUTPUT NAME - File consisting of a "1" or "0" for each neighbor template depending on probabilities calculated. This file will be used in Conditioned Image File creation and restoration.

ORDER OF PARAMETERS:

1) Number of records in input file

2) CCITT File #

3) File Resolution

PROGRAM: CREATE

DESCRIPTION:

This program creates a CIF along with statistics requested in subtask 1 (see Section 2.0). The file generated in CHART is read in and used to generate the CIF.

CALLING SEQUENCE

CREATE,<INPUT NAME>, <OUTPUT NAME>, <PROBABILITY FILE>

INPUT NAME - Input Name of Image

OUTPUT NAME - Output Name of CIF

PROBABILITY FILE - File Generated by CHART

ORDER OF PARAMETERS

)1 # Word per output record

2) # records to be output

3) CCITT file number

4) File resolution

5) # records in input file

PROGRAM: RESTORE

DESCRIPTION:

This program restores the CIF back to the original image file. The probability file used to generate the CIF is read in and used to restore the file to its original state.

CALLING SEQUENCE:

RESTORE - <INPUT NAME>, <OUTPUT NAME>, <PROBABILITY FILE>

INPUT NAME - Input CIF Name

OUTPUT NAME - Restored Output Image File

PROBABILITY FILE - File used to create CIF; file is calculated in program CHART.

ORDER OF PARAMETERS

1) # of words to be output

2) # records to be output

3) # records in input file

PROGRAM: ENCODE

DESCRIPTION:

This program encodes an input CIF into an output encoded file. Before encoding each line, a codebook, as discussed in Section 3.0, is generated and stored. The output file will be written to disk if an output file is requested. There is also an option to print line by line compression statistics.

CALLING SEQUENCE

ENCODE - <INPUT NAME>,<OUTPUT NAME>, <CODE FILE>,
<WEIGHT FILE>

INPUT NAME - Input CIF Name

OUTPUT NAME - Output Encoded File Name

ORDER OF PARAMETERS

1) # records per output file

2) Decision for output file

3) Max weight for segment (Table 3.1)

4) CCITT file number

5) File resolution

6) Arithmetic word length

7) # words per input record

8) # records in input file

9) Segment length to be compressed

10) Decision for line by line compression stats.

PROGRAM: DECODE

DESCRIPTION:

Program DECODE decodes the encoded file back into a CIF. The prestored codebook generated in program encode is read in and used for decoding the segment weights.

A linked Huffman decoding tree is read in which finds the Huffman code used for the segment weight coding. The Huffman code is then looked up in the prestored codebook to find the segment weight. Once the weight of the segment is found the rank can be easily calculated.

CALLING SEQUENCE:

DECODE <INPUT NAME>, <OUTPUT NAME>, <WEIGHT FILES>, <TREE FILES>

INPUT NAME - Encoded Input CIF Name

OUTPUT NAME - Decoded Output CIF name

ORDER OF PARAMETERS

1) # words per output record

2) # records to be output

3) # words per input record

4) # records in input file

5) Segment length to be compressed

6) Max weight encodable (Table 3.1)

APPENDIX C

SOFTWARE CODE LISTINGS

```
$FILES 3,4,5

      PROGRAM CHART

C****************************************** ****************************
C                                                                      *
C     AUTHOR: J. P. DIMAGGIO                                           *
C                                                                      *
C     DESCRIPTION:                                                     *
C                                                                      *
C     This program develops a table of the number of occurrences of   *
C     neighboring bit patterns and the state of the current bit.      *
C     It also develops a probability table and output file with       *
C     respective probabilities of bit patterns and bit states.        *
C                                                                      *
C                                                                      *
C   CALLING SEQUENCE:                                                  *
C                                                                      *
C      RU,CHART,<INPUT NAMR>,<OUTPUT NAMR>                             *
C                                                                      *
Cooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
C                                                                      *
C   INPUT PARAMTERS:                                                   *
C                                                                      *
C                 NUMBER OF RECORDS TO PROCESS                         *
C                 CCITT FILE NUMBER                                    *
C                 FILE RESOLUTION                                      *
C                                                                      *
C**********************************************************************
C                                                                      *
      IMPLICIT NONE
C                                                                      *
C**********************************************************************
C
C  DEFINE PARAMETERS
C
      INTEGER MAXLIN        ! MAXIMUM NO. OF OUTPUT RECORDS
      INTEGER BUFD          ! MAXIMUM NO. OF WORDS PER INPUT RECORD
      INTEGER BITS          ! MAX NO. OF BITS PER RECORD
      INTEGER DERECS        ! HOW MANY RECORDS TO BE PROCESSED

      PARAMETER (BUFD=512)
      PARAMETER (BITS=3500)
C
C************ FILE DEFINITIONS ****************************************
C
      INTEGER   ISTAT              ! I-O STATUS VARIABLE
      INTEGER   LBUF(BUFD)         ! READ/WRITE BUFFER FOR FORTRAN 77
      INTEGER   BUFFER(BUFD)       ! RECORD BUFFER
      INTEGER*4  MREC              ! NO. OF RECORDS IN INPUT FILE
      INTEGER*4 BUFMAX             ! INPUT RECORD SIZE IN WORDS
C
C************* LOCAL VARIABLES ****************************************
C
      INTEGER   PREV(-2:BITS)  ! HOLDS BIT VALUES OF PREVIOUS LINE READ
```

```
      INTEGER   TABLE(16,4)     ! TABLE OF NEIGHBORING BIT PATTERNS
      REAL      PROB(16,2)      ! PROBABILITY ARRAY
      INTEGER*4 COUNT(16,3)     ! RESULT OCCURRENCE ARRAY
      INTEGER*4 TOTTOT          ! TOTAL BIT COUNT
      INTEGER*4 TOTBLK          ! TOTAL BLACK BIT COUNT
      INTEGER*4 TOTWHT          ! TOTAL WHITE BIT COUNT
      INTEGER   NBR(4)          ! HOLDS VALUES OF NEIGHBORING BITS
      INTEGER   BITVAL          ! VALUE OF CURRENT BIT
      INTEGER   PREBIT          ! VALUE OF PREVIOUS BIT
      REAL      BITNUM          ! NUMBER OF BITS IN FILE
      INTEGER   BITWRD          ! PRESENT WORD IN BUFFER
      INTEGER   CT              ! INDEX OF CURRENT BIT IN LINE
      INTEGER   P,I,J,K,L,Q     ! LOOP COUNTERS
      INTEGER   FLNUM           ! CCITT FILE NUMBER
      INTEGER   RESLTN          ! CURRENT FILE  RESOLUTION
      INTEGER   FIND            ! INDEX USED IN SEARCH
      INTEGER   MUL             ! NARROWING VARIABLE USED IN SEARCH
C
C************* FILE PARAMETERS ****************************************
C
      CHARACTER INFILE*20
      CHARACTER ACCTYP*20
      CHARACTER DTFIL*20
      INTEGER   IRCL,OREC         ! FILE PARAMETERS
      INTEGER   ITBUF(15)         ! ARRAY FOR TIME-OF-DAY
      INTEGER   TERM              ! TERMINAL LU
      INTEGER   LFIL              ! PRINTER LU
      INTEGER   OUTFIL            ! OUTPUT FILE LU
      INTEGER   PELFIL            ! LU OF INPUT FILE
      INTEGER   ITLOG             ! LIBRARY FUNCTION
      LOGICAL   EXISTS

      DATA TERM,LFIL,PELFIL,OUTFIL/1,6,7,8/
```

```
C
C*********** INITIALIZING *********************************************
C
      DO I=-2,BITS
         PREV(I)=0
      END DO

      DO I=1,16
         DO J=1,3
            COUNT(I,J)=0
         END DO
      END DO
C
C****************** BEGIN PROGRAM *************************************
C
C    GET INPUT FILE NAME
C
      CALL RCPAR(1,INFILE)
      INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
     &        EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
      IF(ISTAT.NE.0) STOP 'INQUIRE ERROR'
      IF(EXISTS) THEN
         IF(ACCTYP.EQ.'SEQUENTIAL') THEN
            IF(IRCL.EQ.0.AND.MREC.EQ.0) THEN
               OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
               CALL LGBUF(LBUF,BUFD)
               READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(I),I=1,BUFD+1)
               IF(ISTAT.NE.0) THEN
                  IRCL=ITLOG()
                  MREC=MAXLIN
                  BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
                  IF(ISTAT.NE.0) STOP 'BACKSPACE ERROR'
               ELSE
                  STOP 'NO ERROR IS AN ERROR'
               END IF
            ELSE
               STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
            END IF
         ELSE IF(ACCTYP.EQ.'DIRECT') THEN
            OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT'
     &         ,RECL=IRCL,MAXREC=MREC)
         ELSE
            STOP 'ACCTYP UNDEFINED'
         END IF
      ELSE
         STOP 'INPUT FILE DOES NOT EXIST'
      END IF
C
C************ READ IN INPUT PARAMETERS ********************************
C
      WRITE(TERM,*)('HOW MANY RECORDS TO PROCESS?')
      READ(TERM,*) DERECS
      MREC=MIN(MREC,DERECS)
      WRITE(TERM,*) 'CCITT IMAGE NUMBER?'
      READ(TERM,*) FLNUM
```

```
      WRITE(TERM,*) 'FILE RESOLUTION?'
      READ(TERM,*) RESLTN
C
C*********** BEGIN FILE PROCESSING ***********************************
C
      BUFMAX=IRCL/2

      DO I=1,MREC

         CT=1
         PREBIT=0

         CALL LGBUF(LBUF,BUFD)
         READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(P),P=1,BUFMAX)
         IF(ISTAT.NE.0) THEN
            WRITE(LFIL,'(''INPUT READ ERROR, ISTAT='',I4)') ISTAT
            STOP
         END IF

         DO K=1,BUFMAX

            BITWRD=BUFFER(K)

            DO L=15,0,-1

               IF (BTEST(BITWRD,L))THEN
                  BITVAL=1
               ELSE
                  BITVAL=0
               END IF
               DO Q=1,3
                  NBR(Q)=PREV((Q-1)+(CT-1))
               END DO
               NBR(4)=PREBIT
               PREV(CT-1)=PREBIT
               PREBIT=BITVAL
               CT=CT+1
C
C************** INCREMENT PROPER NEIGHBOR TEMPLATE *******************
C
               FIND=1
               MUL=16
               DO P=1,4
                  IF (NBR(P).EQ.1) THEN
                     FIND=FIND+MUL/2
                  END IF
                  MUL=MUL/2
               END DO

               COUNT(FIND,1)=COUNT(FIND,1)+1
               IF (BITVAL.EQ.1) THEN
                  COUNT(FIND,2)=COUNT(FIND,2)+1
               ELSE
                  COUNT(FIND,3)=COUNT(FIND,3)+1
               END IF
```

```

               END DO                ! NUMBER BITS IN WORD

            END DO              ! NUMBER OF WORDS PER RECORD

         PREV(CT-1)=BITVAL

         END DO            ! NUMBER OF RECORDS PER LINE
C
C************* SET UP TABLE ******************************************
C
         MUL=8
         DO J=1,4
            BITVAL=0
            DO I=1,16
               TABLE(I,J)=BITVAL
               IF (MOD(I,MUL).EQ.0) THEN
                  BITVAL=BITVAL.NEQV.1
               END IF
            END DO
            MUL=MUL/2
         END DO
C
C************* GET OUTPUT FILE NAME AND OPEN *************************
C
          CALL RCPAR(2,DTFIL)
          OPEN(UNIT=OUTFIL,FILE=DTFIL)
C
C************* WRITE INPUT PARAMETERS ********************************
C
         CALL FTIME(ITBUF)
         WRITE(LFIL, '(1H0,15A2)') ITBUF
         WRITE(LFIL, '(''0INPUT FILE NAME - '',A20)') INFILE
         WRITE(LFIL, '(''0OUTPUT FILE NAME - '',A20)') DTFIL
         WRITE(LFIL,50) IRCL,MREC,FLNUM,RESLTN
   50    FORMAT('0INPUT PARAMETERS:'/
     *          ' INPUT RECORD SIZE ='I7' BYTES'/
     *          ' NO. OF INPUT RECORDS ='I7/
     *          ' FACSIMILE IMAGE: CCITT IMAGE #'I3/
     *          ' RESOLUTION: 'I4' LINES PER INCH'//)
C
C**************** PRINT HEADINGS *************************************
C
         WRITE(LFIL,*)('A B C D #OCCURRENCES   BLACK     WHITE')
         WRITE(LFIL,*)('* * * * ************   *****     *****')
C
C**************** PRINT COUNT RESULTS ********************************
C
         BITNUM=REAL(MREC*BUFMAX*16)
         TOTTOT=0
         TOTBLK=0
         TOTWHT=0
         DO I=1,16
            WRITE(LFIL,100)((TABLE(I,J),J=1,4),(COUNT(I,J),J=1,3))
            TOTTOT=TOTTOT+COUNT(I,1)
```

```
            TOTBLK=TOTBLK+COUNT(I,2)
            TOTWHT=TOTWHT+COUNT(I,3)
            IF (COUNT(I,1).EQ.0) THEN
               PROB(I,2)=0.0
            ELSE
               PROB(I,2)=REAL(COUNT(I,3))/REAL(COUNT(I,1))
            END IF
            COUNT(I,1)=REAL(COUNT(I,1))
            PROB(I,1)=COUNT(I,1)/BITNUM
         END DO

  100    FORMAT (4(I2),1X,3(I10))
         WRITE(LFIL,125) TOTTOT,TOTBLK,TOTWHT
  125    FORMAT ('------------------------------------------------'/
     *           ' TOTAL    'I10,I10,I10///)
C
C*********** PRINT HEADINGS ******************************************
C
         WRITE(LFIL,*)('                 STATE')
         WRITE(LFIL,*)('A B C D      PROBABILITY    PROB[X=0|STATE]')
         WRITE(LFIL,*)('* * * *      ***********    ***************')
C
C****************** PRINT PROBABILITY RESULTS ***********************
C
         WRITE(LFIL,200)((TABLE(I,J),J=1,4),(PROB(I,J),J=1,2),I=1,16)
  200    FORMAT (16(4(I2),(F14.4),(F17.4)/))
         CALL RCPAR(2,DTFIL)
         OPEN(UNIT=OUTFIL,FILE=DTFIL)
         DO I=1,16
            IF (PROB(I,2).GE.0.5) THEN
               BITVAL=0
               WRITE(OUTFIL,300) (BITVAL)
            ELSE
               BITVAL=1
               WRITE(OUTFIL,300) (BITVAL)
            END IF
         END DO

  300    FORMAT(I3)

C*********************************************************************
C                                                                    *
C        END OF PROGRAM                                              *
C                                                                    *
C*********************************************************************
         END
```

Module CHART   No errors   Prog: 8995   Blank Common: None
FTN7X 2440/850304   No warnings   Save: None   Local Ema: None

```
$FILES 3,4,5
      PROGRAM CREATE

C**********************************************************************
C
C     AUTHOR J. P. DIMAGGIO
C
C     DESCRIPTION:
C
C     This program creates a conditioned image file using
C     the probability data cenerated by PROGRAM CHART.
C     It also generates all necessary statistics of the
C     structure of the CIF.
C
C   CALLING SEQUENCE:
C
C      RU,CREATE,<INPUT NAMR>,<OUTPUT NAMR>,<PARAMETER FILE>,<PROBABILITY FILEC
Coooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
C
C    INPUT PARAMETERS:
C              NUMBER OF WORDS PER OUTPUT RECORD
C              NUMBER OF RECORDS PER OUTPUT FILE
C              NUMBER OF RECORDS IN INPUT FILE
C
C************************************************************************
C
      IMPLICIT NONE
C
C************************************************************************
C
C  DEFINE PARAMETERS
C
      INTEGER MAXLIN       ! MAXIMUM NO. OF OUTPUT RECORDS
      INTEGER DERECS       ! RECORDS IN INPUT FILE
      INTEGER BUFD         ! MAXIMUM NO. OF WORDS PER INPUT RECORD
      INTEGER BITS         ! MAXIMUM NO. OF BITS PER RECORD RECORD

      PARAMETER (MAXLIN=5000)
      PARAMETER (BITS=3500)
      PARAMETER (BUFD=512)
C
C************ FILE DEFINITIONS ******************************************

      INTEGER   ISTAT           ! I-O STATUS VARIABLE
      INTEGER   LBUF(BUFD)      ! READ/WRITE BUFFER FOR FORTRAN 77
      INTEGER   BUFFER(BUFD)    ! RECORD BUFFER

      INTEGER    LINMAX         ! NO. OF RECORDS TO BE OUTPUT
      INTEGER    BUFDIM         ! NO. OF WORDS PER OUTPUT RECORD
      INTEGER*4  MREC           ! NO. OF RECORDS IN INPUT FILE

      INTEGER*4 BUFMAX          ! INPUT RECORD SIZE IN WORDS
C
C************* LOCAL VARIABLES *****************************************
C
```

```
      INTEGER    PREV(-1:BITS)        ! HOLDS BIT VALUES OF PREVIOUS LINE
      INTEGER    WEIGHT(0:BITS)       ! HOLDS WEIGHTS
      INTEGER*4  RNLGTH(0:BITS)       ! HOLDS RUNLENGTHS OF ENTIRE FILE
      INTEGER*4  THWT(0:32)           ! HISTOGRAM DATA FOR 32 BITS IN ROW
      INTEGER*4  SXWT(0:64)           ! HISTOGRAM DATA FOR 64 BITS IN ROW
      INTEGER*4  TOTAL                ! HOLDS TOTALS FOR HISTOGRAMS
      INTEGER    SIXCT                ! WEIGHT COUNT FOR 32 BIT SEGMENTS
      INTEGER    THRCT                ! WEIGHT COUNT FOR 64 BIT SEGMENTS
      INTEGER    PROB(16)             ! DATA TO BE USED IN CALCULATIONS
      INTEGER    NBR(4)               ! HOLDS VALUES OF NEIGHBORING BITS
      INTEGER    BITVAL               ! HOLDS VALUE OF CURRENT BIT
      INTEGER    PREBIT               ! HOLDS VALUE OF PREVIOUS BIT
      INTEGER*4  BITNUM               ! NUMBER OF BITS IN FILE
      INTEGER    PSHOLD               ! HOLDS POSITION OF 1 BIT IN CSL
      INTEGER*4  LNWGT                ! HOLDS WEIGHT OF EACH CSL
      INTEGER    FLNUM,RESLTN         ! CCITT FILE INFORMATION
      INTEGER*4  FIWGT                ! HOLDS VALUE OF WEIGHT OF FILE
      INTEGER    OBUFF(BUFD)          ! OUTPUT BUFFER
      INTEGER    BITWRD               ! PRESENT WORD IN INPUT BUFFER
      INTEGER    OBTWRD               ! PRESENT WORD IN OUTPUT BUFFER
      INTEGER    CT                   ! INDEX OF CURRENT BIT IN LINE
      INTEGER    P,I,K,L,Q            ! LOOP COUNTERS
      INTEGER    FIND                 ! INDEX USED IN SEARCH
      INTEGER    MUL                  ! NARROWING VARIABLE USED IN SEARCH
C
C**************FILE PARAMETERS*******************************
C
      CHARACTER INFILE*20
      CHARACTER OTFILE*20
      CHARACTER DATFIL*20
      CHARACTER ACCTYP*20
      INTEGER    IRCL,JRCL,OREC   ! FILE PARAMETERS
      INTEGER    ITBUF(15)        ! ARRAY FOR TIME-OF-DAY
      INTEGER    TERM             ! TERMINAL LU
      INTEGER    LPFIL            ! PRINTER LU
      INTEGER    PELFIL           ! LU OF INPUT FILE
      INTEGER    OUTFIL           ! LU OF OUTPUT FILE
      INTEGER    DTFIL            ! LU OF DATA FILE
      INTEGER    ITLOG            ! LIBRARY FUNCTION

      LOGICAL    EXISTS

      DATA TERM,LPFIL,PELFIL,OUTFIL,DTFIL/23,6,2,3,4/
```

```
C
C********** INITIALIZING ***********************************************
C
      DO I=-1,BITS
         PREV(I)=0
      END DO

      DO I=0,BITS
         WEIGHT(I)=0
         RNLGTH(I)=0
      END DO

      DO I=0,32
         THWT(I)=0
      END DO

      DO I=0,64
         SXWT(I)=0
      END DO
C
C****************** BEGIN PROGRAM *************************************
C
C    GET INPUT FILE NAME
C
      CALL RCPAR(1,INFILE)
      INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
     &        EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
      IF(ISTAT.NE.0) STOP 'INQUIRE ERROR'
      IF(EXISTS) THEN
         IF(ACCTYP.EQ.'SEQUENTIAL') THEN
            IF(IRCL.EQ.0.AND.MREC.EQ.0) THEN
               OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
               CALL LGBUF(LBUF,BUFD)
               READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(I),I=1,BUFD+1)
               IF(ISTAT.NE.0) THEN
                  IRCL=:TLOG()
                  MREC=MAXLIN
                  BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
                  IF(ISTAT.NE.0) STOP 'BACKSPACE ERROR'
               ELSE
                  STOP 'NO ERROR IS AN ERROR'
               END IF
            ELSE
               STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
            END IF
         ELSE IF(ACCTYP.EQ.'DIRECT') THEN
            OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT'
     &          ,RECL=IRCL,MAXREC=MREC)
         ELSE
            STOP 'ACCTYP UNDEFINED'
         END IF
      ELSE
         STOP 'INPUT FILE DOES NOT EXIST'
      END IF
C
```

```
C*************** READ IN INPUT PARAMETERS ******************************
C
      BUFDIM=0
      DO WHILE (BUFDIM.LT.1.OR.BUFDIM.GT.BUFD)
        WRITE(TERM,'(''$ENTER NUMBER OF WORDS PER OUTPUT RECORD: '')')
        READ(TERM,*) BUFDIM
      END DO

      LINMAX=0
      DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
        WRITE(TERM,'(''$NUMBER OF RECORDS TO BE OUTPUT = :'')')
        READ(TERM,*) LINMAX
      END DO

      WRITE(TERM,*) 'CCITT FILE NUMBER?'
      READ(TERM,*) FLNUM
      WRITE(TERM,*) 'FILE RESOLUTION?'
      READ(TERM,*)   RESLTN
C
C***************** OPEN OUTPUT FILE ************************************
C
      CALL RCPAR(3,OTFILE)
      OREC=MIN0(MREC,LINMAX)
      JRCL=BUFDIM*2
      OPEN(UNIT=OUTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRCL
     &     ,MAXREC=OREC,STATUS='OLD',IOSTAT=ISTAT)
      IF(ISTAT.NE.0) STOP 'OUTPUT FILE OPEN ERROR'

      WRITE(TERM,*)  'NUMBER OF RECORDS IN INPUT FILE?'
      READ(TERM,*) DERECS
      MREC=MIN(MREC,DERECS)
C
C************* WRITE INPUT PARAMETERS **********************************
C
      CALL FTIME(ITBUF)
      WRITE(LPFIL,'(1H0,15A2)') ITBUF
      WRITE(LPFIL,'(''0INPUT FILE NAME - '',A20)') INFILE
      WRITE(LPFIL,'(''0OUTPUT FILE NAME - '',A20)') OTFILE
      WRITE(LPFIL,200) FLNUM,RESLTN,BUFDIM,OREC,IRCL,MREC,JRCL
  200 FORMAT('0INPUT PARAMETERS:'/
     *       '*******************'/
     *       ' FACSIMILE IMAGE: CCITT IMAGE #',I2/
     *       ' RESOLUTION: ',I4,' LINES PER INCH'/
     *       ' NUMBER OF WORDS PER OUTPUT RECORD =',I6/
     *       ' NUMBER OF RECORDS TO BE WRITTEN =',I6/
     *       ' INPUT RECORD SIZE ='I7' BYTES'/
     *       ' NO. OF INPUT RECORDS ='I7/
     *       ' OUTPUT RECORD SIZE ='I7' BYTES')
C
C*********** READ IN PROBABILITY DATA **********************************
C
       CALL RCPAR(4,DATFIL)
       OPEN(UNIT=DTFIL,FILE=DATFIL)
       DO I=1,16
         READ(DTFIL,*) PROB(I)
```

```
        END DO

      BUFMAX=IRCL/2
      FIWGT=0

C
C*********** BEGIN FILE PROCESSING ***********************************
C
      DO I=1,MREC

         CT=1
         PSHOLD=0
         LNWGT=0
         SIXCT=0
         THRCT=0
         PREBIT=0

         CALL LGBUF(LBUF,BUFD)
         READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(P),P=1,BUFMAX)
         IF(ISTAT.NE.0) THEN
           WRITE(TERM,'(''INPUT READ ERROR, ISTAT='',I4)') ISTAT
           STOP
         END IF

         DO K=1,BUFMAX
            BITWRD=BUFFER(K)
            OBTWRD=0

            DO L=15,0,-1
               IF (BTEST(BITWRD,L))THEN
                  BITVAL=1
               ELSE
                  BITVAL=0
               END IF
               DO Q=1,3
                  NBR(Q)=PREV((Q-1)+(CT-1))
               END DO
               NBR(4)=PREBIT
               PREV(CT-1)=PREBIT
               PREBIT=BITVAL
C
C************** FIND PROPER NEIGHBOR TEMPLATE ************************
C
               FIND=1
               MUL=16
               DO P=1,4
                  IF (NBR(P).EQ.1) THEN
                     FIND=FIND+MUL/2
                  END IF
                  MUL=MUL/2
               END DO

               IF (PROB(FIND).NE.BITVAL) THEN
                  OBTWRD=IBSET(OBTWRD,L)
                  LNWGT=LNWGT+1
```

```
                    SIXCT=SIXCT+1
                    THRCT=THRCT+1
                    RNLGTH(CT-PSHOLD)=RNLGTH(CT-PSHOLD)+1
                    PSHOLD=CT
                 END IF
                 IF (MOD(CT,32).EQ.Ø) THEN
                    THWT(THRCT)=THWT(THRCT)+1
                    THRCT=Ø
                 END IF
                 IF (MOD(CT,64).EQ.Ø) THEN
                    SXWT(SIXCT)=SXWT(SIXCT)+1
                    SIXCT=Ø
                 END IF
                 CT=CT+1
              END DO      ! NO BITS IN WORD

              OBUFF(K)=OBTWRD

           END DO    ! NO WORDS IN RECORD

           PREV(CT-1)=BITVAL
           FIWGT=FIWGT+LNWGT
           WEIGHT(LNWGT)=WEIGHT(LNWGT)+1
C
C************* WRITE RECORD TO OUTPUT FILE ***************************
C
           CALL LGBUF(LBUF,BUFD)
           WRITE(UNIT=OUTFIL,IOSTAT=ISTAT) (OBUFF(Q),Q=1,BUFDIM)
           IF (ISTAT.NE.Ø) THEN
              WRITE(TERM,'(''OUTPUT READ ERROR, ISTAT='',I4)') ISTAT
              STOP
           END IF

        END DO ! RECORDS IN FILE
C
C************ PRINT HISTOGRAM ****************************************
C
        BITNUM=MREC*16*BUFMAX
        WRITE(LPFIL,*) ' '
        WRITE(LPFIL,25Ø) BITNUM,FIWGT
 25Ø    FORMAT ('ØNUMBER OF BITS IN CIF  T=',I12/
     *          ' WEIGHT  OF  CIF   W=      ',I12//)
        WRITE(LPFIL,*) 'HISTOGRAM OF WEIGHTS OF CONDITIONED SCAN'
        WRITE(LPFIL,*) 'LINES WITHIN THE CONDITIONED IMAGE FILE'
        WRITE(LPFIL,*) ' '
        WRITE(LPFIL,*) 'WEIGHT            ØLINES'
        TOTAL=Ø
        WRITE(LPFIL,*) '******            ******'

        DO I=Ø,BITS
          IF(WEIGHT(I).NE.Ø) THEN
             WRITE(LPFIL,3ØØ) (I,WEIGHT(I))
             TOTAL=TOTAL+WEIGHT(I)
          END IF
        END DO
```

```
      WRITE(LPFIL,270) TOTAL

      WRITE(LPFIL,*) 'RUNLENGTHS IN CONDITIONED IMAGE FILE'
      WRITE(LPFIL,*) ' '
      WRITE(LPFIL,*) 'RUNLENGTHS   #OCCURRENCES'
      TOTAL=0
      WRITE(LPFIL,*) '**********   ************'

      DO I=1,BITS
        IF(RNLGTH(I).NE.0) THEN
           WRITE(LPFIL,300) (I,RNLGTH(I))
           TOTAL=TOTAL+RNLGTH(I)
        END IF
      END DO

      WRITE(LPFIL,270) TOTAL
      WRITE(LPFIL,*) 'THIRTY TWO BIT SEGMENT HISTOGRAM'
      WRITE(LPFIL,*) ' '
      WRITE(LPFIL,*) 'WEIGHT       #OCCURRENCES'
      TOTAL=0
      WRITE(LPFIL,*) '******       ************'

      DO I=0,32
        IF (THWT(I).NE.0) THEN
           WRITE(LPFIL,300) (I,THWT(I))
           TOTAL=TOTAL+THWT(I)
        END IF
      END DO

      WRITE(LPFIL,270) TOTAL
      WRITE(LPFIL,*) 'SIXTY-FOUR BIT SEGMENT HISTOGRAM'
      WRITE(LPFIL,*) ' '
      WRITE(LPFIL,*) 'WEIGHT       #OCCURRENCES'
      TOTAL=0
      WRITE(LPFIL,*) '******       ************'

      DO I=0,64
        IF (SXWT(I).NE.0) THEN
           WRITE(LPFIL,300) (I,SXWT(I))
           TOTAL=TOTAL+SXWT(I)
        END IF
      END DO

      WRITE(LPFIL,270) TOTAL
  270 FORMAT ('--------------------------------------------',/
     *        ' TOTAL',I19//
     *        '********************************************'/)
  300 FORMAT (I5,I20)

      CLOSE(UNIT=PELFIL)
      CLOSE(UNIT=OUTFIL)

C*****************************************************************
C                                                                *
C        END OF PROGRAM                                          *
```

```
C                                                                    *
C*********************************************************************

      END
```

```
Module CREATE           No errors     Prog: 20671    Blank Common: None
FTN7X  2440/850304      No warnings   Save:  None    Local Ema:    None
```

```
$FILES 2,2

      PROGRAM ENCODE

C*****************************************************************
C                                                                *
C     AUTHOR J. P. DIMAGGIO                                      *
C                                                                *
C     DESCRIPTION:                                               *
C                                                                *
C     This program encodes a conditioned image file             *
C     using SLDC encoding techniques. The user is               *
C     prompted for all input parameters including               *
C     file sizes, amount of file to process, input              *
C     design parameters, and option to create an                *
C     output file (and/or) print line by line                   *
C     compression statistics.                                    *
C                                                                *
C                                                                *
C                                                                *
C   CALLING SEQUENCE:                                            *
C                                                                *
C      RU,ENCODE,<INPUT NAMR>,<OUTPUT NAMR>                      *
C                                                                *

Cooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo*
C                                                                *
C   INPUT PARAMETERS:                                            *
C              NUMBER OF RECORDS PER OUTPUT FILE                 *
C              DECISION FOR OUTPUT FILE                          *
C              MAXIMUM WEIGHT PER SEGMENT                        *
C              CCITT FILE NUMBER                                 *
C              FILE RESOLUTION                                   *
C              ARITHMETIC WORD LENGTH                            *
C              NUMBER OF WORDS PER INPUT FILE                    *
C              NUMBER OF RECORDS IN INPUT FILE                   *
C              SEGMENT LENGTH TO BE COMPRESSED                   *
C              DECISION FOR LINE BY LINE COMPRESSION STATS       *
C                                                                *
C*****************************************************************


      INCLUDE ENCODE.INC
C*******************************************************************
C
      IMPLICIT NONE
C
C*******************************************************************
C
C  DEFINE PARAMETERS
C
      INTEGER MAXLIN       ! MAXIMUM NO. OF OUTPUT RECORDS
      INTEGER DERECS       ! RECORDS IN INPUT FILE
      INTEGER BUFD         ! MAXIMUM NO. OF WORDS PER INPUT RECORD
C
      PARAMETER (MAXLIN=5000)
```

```
      PARAMETER (BUFD=225)
C
C*************FILE DEFINITIONS********************************
      INTEGER   ISTAT             ! I-O STATUS VARIABLE
      INTEGER   LBUF(BUFD)        ! READ/WRITE BUFFER FOR FORTRAN 77
      INTEGER   CSL(BUFD)         ! CSL BUFFER
      INTEGER   ESL(BUFD)         ! ENCODED SCAN LINE BUFFER
C
      INTEGER    LINMAX  ! NO. OF RECORDS TO BE OUTPUT
      INTEGER    BUFDIM  ! NO. OF WORDS PER OUTPUT RECORD
      INTEGER*4  MREC    ! NO. OF RECORDS IN INPUT FILE
C
      INTEGER   BUFMAX                  ! INPUT RECORD SIZE IN WORDS
C
C**************LOCAL VARIABLES********************************
C
      INTEGER*4 WEIGHT(-1:34)       ! HOLDS HUFFMAN CODES IN PROPER ORDER
      INTEGER   CODLEN(-1:34)       ! HOLDS HUFF. CODE WEIGHTS IN PROP. ORDER
      INTEGER   SEGBUF(BUFD)        ! SEGMENT BUFFER
      INTEGER   MAXWGT              ! MAX WEIGHT ENCODABLE OFR SEGMENT
      INTEGER   CTWRD               ! CURRENT WORD IN CSL
      INTEGER   ESLPOS              ! CURRENT POSITION IN ESL
      INTEGER   CSLPOS              ! CURRENT POSITION IN CSL
      REAL      RATIO               ! COMPRESSION RATIO
      INTEGER   NBPW                ! NUMBER OF BITS PER WORD
      INTEGER   CHECK               ! BIT HOLDER
      INTEGER*4 TOTAL               ! TOTAL NUMBER OF BITS COMPRESSED
      INTEGER   WRDS                ! NUMBER OF WORDS IN SEGMENT
      INTEGER   SEGS                ! NUMBER OF SEGMENTS IN A LINE
      INTEGER   SEGCT               ! CURRENT COUNT OF SEGMENTS IN A LINE
      INTEGER   FLNUM               ! CCITT IMAGE FILE NUMBER
      INTEGER   RESLTN              ! FILE RESOLUTION
      INTEGER*4 CODE                ! HUFFMAN CODE OF SEGMENT WEIGHT
      INTEGER   SEGWGT              ! CURRENT SEGMENT WEIGHT
      INTEGER   POSITN              ! CURRENT POSITION IN SEGMENT
      INTEGER*4 RANK                ! RANK OF SEGMENT
      INTEGER   SGONES              ! ONES SEEN IN SEGMENT
      INTEGER*4 R1,R2               ! TEMPORARIES IN CASE OF 32 BIT WORD
      INTEGER   L1                  !   " "                 " "
      INTEGER   LNONES              ! ONES SEEN IN LINE
      INTEGER   LEN                 ! LENGTH OF RANK
      REAL      LOG2                ! CONSTANT USED IN EQUATION
      INTEGER   NUMBER              ! NUMBER OF BITS IN A LINE
      INTEGER*4 BINOM               ! BINOMIAL FUNCTION RETURN
      INTEGER*4 LNWGT               ! HOLDS WEIGHT OF EACH CSL
      INTEGER   TMWGT               ! SINGLE INTEGER VALUE FOR LNWGT
      INTEGER   SEGL                ! SEGMENT LENGTH TO BE COMPRESSED
      INTEGER   BITWRD              ! PRESENT WORD IN INPUT BUFFER
      INTEGER   CT                  ! INDEX OF CURRENT BIT IN LINE
      INTEGER   M,I,K,L,Q           ! LOOP COUNTERS
      CHARACTER DECIDE              ! DECISION FLAG FOR OUTPUT FILE
      CHARACTER PRISTS              ! DECISION FLAG FOR LINE BY LINE
                                    ! COMPRESSION STATISTICS
C
C**************FILE PARAMETERS********************************
```

```
C
      CHARACTER INFILE*20
      CHARACTER OTFILE*20
      CHARACTER ACCTYP*20
      INTEGER   IRCL,JRCL,OREC   ! FILE PARAMETERS
      INTEGER   ITBUF(15)        ! ARRAY FOR TIME-OF-DAY
      INTEGER   TERM             ! TERMINAL LU
      INTEGER   LPFIL            ! PRINTER LU
      INTEGER   PELFIL           ! LU OF INPUT FILE
      INTEGER   OUTFIL           ! LU OF OUTPUT FILE
      INTEGER   ITLOG            ! LIBRARY FUNCTION
C
      LOGICAL   EXISTS
      LOGICAL   DONSEG           ! TRUE IF SEG IS FINISHED ENCODING
      LOGICAL   DONLIN           ! TRUE IF LINE IS FINISHED ENCODING
      EXTERNAL  BINOM            ! FUNCTION USED TO COMPUTE BINOMIAL
C                                  COEFFICIENT
      EXTERNAL  MB4B             ! DESCRIBED IN REPORT
      EXTERNAL  MI2B             !  "           "

      COMMON  /ENCOD/WEIGHT,SEGL,NUMBER,CODLEN
C
C************INITIALIZING****************************
C
      DATA TERM,LPFIL,PELFIL,OUTFIL/1,6,2,3/
```

```
C*********************************************************************
C                                                                    *
C     BEGIN PROGRAM                                                  *
C                                                                    *
C*********************************************************************

      DO L=0,33
        WEIGHT(L)=0
      END DO

      LOG2=LOG(2.0)
C
C   GET INPUT FILE NAME
C
      CALL RCPAR(1,INFILE)
      INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
     &        EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
      IF(ISTAT.NE.0) STOP 'INQUIRE ERROR'
      IF(EXISTS) THEN
         IF(ACCTYP.EQ.'SEQUENTIAL') THEN
            IF(IRCL.EQ.0.AND.MREC.EQ.0) THEN
               OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
               CALL LGBUF(LBUF,BUFD)
               READ(UNIT=PELFIL,IOSTAT=ISTAT) (CSL(I),I=1,BUFD+1)
               IF(ISTAT.NE.0) THEN
                  IRCL=ITLOG()
                  MREC=MAXLIN
                  BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
                  IF(ISTAT.NE.0) STOP 'BACKSPACE ERROR'
               ELSE
                  STOP 'NO ERROR IS AN ERROR'
               END IF
            ELSE
               STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
            END IF
         ELSE IF(ACCTYP.EQ.'DIRECT') THEN
            OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT'
     &           ,RECL=IRCL,MAXREC=MREC)
         ELSE
            STOP 'ACCTYP UNDEFINED'
         END IF
      ELSE
         STOP 'INPUT FILE DOES NOT EXIST'
      END IF

      BUFDIM=IRCL/2
C
C   Read number of records to be written
C
      LINMAX=0
      DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
        WRITE(TERM,*)' HOW MANY RECORDS ARE TO BE PROCESSED?'
        READ(TERM,*) LINMAX
      END DO
      DECIDE = 'X'
```

```
      DO WHILE (DECIDE.NE.'Y'.AND.DECIDE.NE.'N')
         WRITE(TERM,*) 'CREATE OUTPUT FILE? (Y or N)'
         READ(TERM,'(A1)') DECIDE
      END DO
C
C  Get output file name and open if requested
C

      IF(DECIDE.EQ.'Y') THEN
         CALL RCPAR(2,OTFILE)
         OREC=MIN0(MREC,LINMAX)
         JRCL=BUFDIM*2
         OPEN(UNIT=OUTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRCL
     &     ,MAXREC=OREC,STATUS='NEW',IOSTAT=ISTAT)
         IF(ISTAT.NE.0) THEN
             WRITE(TERM,*) 'OUTPUT FILE OPEN ERROR',ISTAT
             STOP
         END IF
      END IF
C
C  Inquire for other input parameters
C
      WRITE(TERM,*) 'MAXIMUM WEIGHT PER SEGMENT?'
      READ(TERM,*) MAXWGT
      WRITE(TERM,*) 'CCITT FILE NUMBER?'
      READ(TERM,*) FLNUM
      WRITE(TERM,*) 'RESOLUTION?'
      READ(TERM,*) RESLTN
      WRITE(TERM,*) 'ARITHMETIC WORD LENGTH?'
      READ(TERM,*) NBPW
      WRITE(TERM,*) 'WORDS PER INPUT RECORD?'
      READ(TERM,*) BUFMAX
      WRITE(TERM,*) 'NO. OF INPUT RECORDS?'
      READ(TERM,*) DERECS
      WRITE(TERM,*) 'SEGMENT LENGTH PARAMETER?'
      READ(TERM,*) SEGL
      MREC=MIN(MREC,DERECS)
C
C   Write input parameters
C
      WRITE(LPFIL,*) ' SLDC ENCODING PROGRAM '
      WRITE(LPFIL,*) ' ********************* '
      WRITE(LPFIL,*) ' '
C
      CALL FTIME(ITBUF)
      WRITE(LPFIL,'(1H0,15A2)') ITBUF
      WRITE(LPFIL,'('' INPUT FILE NAME - '',A20)') INFILE
      IF (DECIDE.EQ.'Y') THEN
         WRITE(LPFIL,'('' OUTPUT FILE NAME - '',A20)') OTFILE
      END IF
C
      WRITE(LPFIL,200) IRCL,MREC,JRCL,OREC,NBPW,FLNUM,RESLTN
  200 FORMAT(' INPUT PARAMETERS:'/
     *       ' *****************'/
     *       ' INPUT RECORD SIZE ='I7' BYTES'/
```

```
     *          ' NO. OF INPUT RECORDS ='I6/
     *          ' OUTPUT RECORD SIZE ='I6' BYTES'/
     *          ' NO. OF OUTPUT RECORDS ='I5/
     *          ' NUMBER OF BITS PER WORD=',I3//
     *          ' COMPRESSED CIF FROM: CCITT IMAGE # 'I2/
     *          ' RESOLUTION: ',I4,' LINES PER INCH'/)


      WRDS=SEGL/16
      NUMBER=BUFMAX*16
      SEGS=NUMBER/SEGL+1
      WRITE(LPFIL,*) 'SEGMENT LENGTH BEING COMPRESSED= ',SEGL
      WRITE(LPFIL,*) 'MAXIMUM WEIGHT ENCODABLE= ',MAXWGT

      PRISTS = 'X'
      DO WHILE(PRISTS.NE.'Y'.AND.PRISTS.NE.'N')
         WRITE(TERM,*) 'LINE BY LINE COMPRESSION STATISTICS? (Y or N)'
         READ(TERM,'(A1)') PRISTS
      END DO

      IF (PRISTS.EQ.'Y') THEN
         WRITE(LPFIL,*) ' COMPRESSION STATISTICS FOR EACH LINE'
         WRITE(LPFIL,*) ' *************************************'
         WRITE(LPFIL,*) ' LINE   BITS   COMPRESSION'
      END IF
C
C************** BEGIN FILE ENCODING **********************************
C
      TOTAL=0
      DO Q=1,MREC
         DO M=1,BUFMAX
            ESL(M)=0
            CSL(M)=0
         END DO
         CALL LGBUF(LBUF,BUFD)
         READ(UNIT=PELFIL,IOSTAT=ISTAT)(CSL(M),M=1,BUFMAX)
         IF(ISTAT.NE.0) THEN
            WRITE(TERM,'(''INPUT READ ERROR ,ISTAT='',I4)') ISTAT
            STOP
         END IF
         LNWGT=0
         CSLPOS=1
C
C****** Calculate line weight ******************************************
C
         DO L=1,BUFMAX
            BITWRD=CSL(L)
            DO K=15,0,-1
               IF(BTEST(BITWRD,K)) THEN
                  LNWGT=LNWGT+1
               END IF
            END DO
         END DO
         IF(LNWGT.EQ.0) THEN
            ESL(1)=IBSET(ESL(1),15)
```

```
             TOTAL=TOTAL+1
             ESLPOS=1
          ELSE
            CALL MI2B(LNWGT,ESL,1,13)
            ESLPOS=14
            TMWGT=LNWGT
            CALL CODGEN(TMWGT,MAXWGT)
            DONLIN=.FALSE.
            CTWRD=1
            LNONES=0
            SEGCT=0
            DO WHILE (.NOT.DONLIN)
C
C*** Begin line encoding *************************************************
C
               CALL MB4B(SEGBUF,1,SEGL,CSL,CSLPOS)
               CSLPOS=CSLPOS+SEGL
               SEGCT=SEGCT+1
               SEGWGT=0
               CT=0
               M=0
               DO WHILE(CT.LT.SEGL)
                  IF(MOD(CT,16).EQ.0) THEN
                     M=M+1
                     CHECK=SEGBUF(M)
                     K=16
                  END IF
                  K=K-1
                  CT=CT+1
                  IF(BTEST(CHECK,K)) THEN
                     SEGWGT=SEGWGT+1
                  END IF
               END DO
               LEN=CODLEN(SEGWGT)
               CODE=WEIGHT(SEGWGT)
               LNONES=LNONES+SEGWGT
C
C**** Encode line weight *************************************************
C
               IF((SEGWGT.GT.MAXWGT).OR.(SEGWGT.EQ.0)) THEN
                  IF(SEGWGT.EQ.0) THEN
                     CALL MI2B(CODE,ESL,ESLPOS,LEN)
                     ESLPOS=ESLPOS+LEN
                  ELSE
                     CODE=WEIGHT(MAXWGT+1)
                     LEN=CODLEN(MAXWGT+1)
                     CALL MI2B(CODE,ESL,ESLPOS,LEN)
                     ESLPOS=ESLPOS+LEN
                     CALL MB4B(ESL,ESLPOS,SEGL,SEGBUF,1)
                     ESLPOS=ESLPOS+SEGL
                 END IF
               ELSE
                  CALL MI2B(CODE,ESL,ESLPOS,LEN)
                  ESLPOS=ESLPOS+LEN
                  DONSEG=.FALSE.
```

```
                POSITN=SEGL
                RANK=0
                M=0
                CT=16
                SGONES=0
                DO WHILE(.NOT.DONSEG)
C
C*********** Encode rank ***********************************************
C
                   M=M+1
                   CHECK=SEGBUF(M)
                   K=16
                   DO WHILE((K.GT.0).AND.(.NOT.DONSEG))
                      K=K-1
                      POSITN=POSITN-1
                      IF((SEGWGT-SGONES).EQ.(POSITN+1)) THEN
                         DONSEG=.TRUE.
                      ELSE
                         IF(BTEST(CHECK,K)) THEN
                            IF(POSITN.NE.0) THEN
                               RANK=RANK+BINOM(POSITN,(SEGWGT-SGONES))
                            END IF
                            SGONES=SGONES+1
                         END IF
                         IF((SGONES.GE.SEGWGT).OR.(POSITN.LE.0)) THEN
                            DONSEG=.TRUE.
                         END IF
                      END IF
                   END DO
                END DO
                LEN=NINT((LOG(REAL(BINOM(SEGL,SEGWGT)))/
     &              LOG2)+0.5)
                IF ((NBPW.EQ.32).AND.(LEN.GT.16)) THEN
                   R1=0
                   L1=MOD(LEN,16)
                   CALL MVBITS(RANK,16,L1,R1,0)
                   LEN=LEN-L1
                   CALL MI2B(R1,ESL,ESLPOS,L1)
                   ESLPOS=ESLPOS+L1
                END IF
                R2=RANK
                CALL MI2B(R2,ESL,ESLPOS,LEN)
                ESLPOS=ESLPOS+LEN
             END IF
             IF((LNONES.GE.LNWGT).OR.(SEGCT.EQ.SEGS)) THEN
                DONLIN=.TRUE.
             END IF
          END DO
          TOTAL=TOTAL+(ESLPOS-1)
         END IF
         IF(DECIDE.EQ.'Y') THEN
            CALL LGBUF(LBUF,BUFD)
            WRITE(UNIT=OUTFIL,IOSTAT=ISTAT) (ESL(M),M=1,BUFMAX)
         END IF
         IF (PRISTS.EQ.'Y') THEN
```

```
               RATIO=REAL(NUMBER)/REAL(ESLPOS)
               WRITE(1,31Ø) Q,ESLPOS,RATIO
            END IF
          END DO
          RATIO=MREC*BUFMAX*16/REAL(TOTAL)
          CALL FTIME(ITBUF)
          WRITE(LPFIL,*) ' FINISHED:'
          WRITE(LPFIL,'(1HØ,15A2)') ITBUF
          WRITE(LPFIL,35Ø) RATIO
 31Ø    FORMAT(I6,I7,F12.2,':1')
 35Ø    FORMAT(/' COMPRESSION RATIO= ',F8.2,':1'/)

C***********************************************************************
C                                                                      *
C       END OF PROGRAM                                                 *
C                                                                      *
C***********************************************************************
        END
```

```
Module ENCODE          No errors      Prog:  3331     Blank Common: None
FTN7X  244Ø/85Ø3Ø4     No warnings    Save:  None     Local Ema:    None
```

```
      SUBROUTINE CODGEN(LNWGT,MAXWGT)

      IMPLICIT NONE

C************* PARAMETER DEFINITION ***********************************
C                                                                    *
      INTEGER  MAXWGT  ! MAXIMUM WEIGHT VALUE ENCODABLE
      INTEGER  LNWGT   ! CURRENT LINE WEIGHT
C                                                                    *
C*********************************************************************
      INTEGER*4 WEIGHT(-1:34)        ! HUFFMAN CODE WEIGHT ARRAY
      REAL*8    PROB(-1:34)          ! DATA TO BE USED IN CALCULATIONS
      INTEGER   INFO(-1:34)          ! HOLDS WEIGHTS OF PROBABILITIES
      REAL*8    NB                   ! REAL VALUE OF BITS PER LINE
      INTEGER   NUMBPL               ! NUMBER OF BITS IN A LINE
      INTEGER*4 BINOM                ! BINOMIAL FUNCTION RETURN
      INTEGER*4 CODE(-1:34)          ! TABLE HOLDING HUFFMAN CODES
      INTEGER   CODLEN(-1:34)        ! CORRESPONDING CODE LENGTHS FOR CODES
      INTEGER   CDWLEN(-1:34)        ! CODE LENGTH ARRAY
      REAL*8    P                    ! PROBABILITY OF 1 ON THAT LINE
      REAL*8    TEMP                 ! USED IN SWAP IN SORT
      INTEGER   TEMP2                ! "      "      "
      INTEGER   SEGL                 ! SEGMENT LENGTH TO BE COMPRESSED
      REAL*8    SUM                  ! USED IN COMPUTING SK'
      INTEGER   MAXWT                ! TEMPORARY MAXWGT VARIABLE
      INTEGER   M,I,Q,J              ! LOOP COUNTERS

      EXTERNAL  BINOM                ! BINOMIAL FUNCTION

      COMMON  /HUFBLK/PROB,CODE,MAXWT,CDWLEN
      COMMON  /ENCOD/WEIGHT,SEGL,NUMBPL,CODLEN
```

```
C**********************************************************************
C                                                                     *
C     BEGIN SUBROUTINE                                                *
C                                                                     *
C**********************************************************************

      DO I=0,33
         CODE(I) = 0
         PROB(I) = 0
      END DO
      MAXWT = MAXWGT
      NB=REAL(NUMBPL)
      SUM=0.0
      IF(REAL(LNWGT).EQ.NB) THEN
         P=0.9999
      ELSE
         P=REAL(LNWGT)/NB
      END IF
      DO J=0,MAXWGT

         PROB(J)=(BINOM(SEGL,J))*(P**J)*((1.0-P)**(SEGL-J))
         INFO(J)=J
         SUM=SUM+PROB(J)

      END DO
      IF ((1.0 - SUM).LT.0) THEN
         PROB(MAXWGT+1)=0.0
      ELSE
         PROB(MAXWGT+1)=1.0-SUM
      END IF
      INFO(MAXWGT+1)=MAXWGT+1
C     WRITE(1,*)(PROB(M),M=0,MAXWGT+1)
C
C**********SORT**********************
C
      DO M=0,MAXWGT+1

         DO Q=MAXWGT+1,M+1,-1

            IF (PROB(Q-1).LT.PROB(Q)) THEN
               TEMP=PROB(Q-1)
               TEMP2=INFO(Q-1)
               PROB(Q-1)=PROB(Q)
               INFO(Q-1)=INFO(Q)
               PROB(Q)=TEMP
               INFO(Q)=TEMP2
            END IF

         END DO

      END DO

      CALL HUFCOD
C
      DO M=0,MAXWGT+1
```

```
          CODLEN(INFO(M))=CDWLEN(M)
          WEIGHT(INFO(M))=CODE(M)
       END DO

       RETURN
C**********************************************************************
C                                                                     *
C      END OF SUBROUTINE                                              *
C                                                                     *
C**********************************************************************
       END
```

```
Module CODGEN          No errors     Prog:   382    Blank Common: None
FTN7X  2440/850304     No warnings   Save:   None   Local Ema:    None
```

```
**********************************************************************
*                                                                    *
*       MODULE NAME: HUFCOD                                          *
*                                                                    *
*                                                                    *
*       DESCRIPTION:  This module generates Huffman codewords        *
*                     and code lengths from probabilities            *
*                     generated by CODGEN.                           *
*                                                                    *
*                                                                    *
*                                                                    *
**********************************************************************

      SUBROUTINE HUFCOD
      IMPLICIT NONE
      INCLUDE HUFFMAN.INC
*     -------  ------- ---

      INTEGER       TBLNDX                          ! Huffman Table Index

      INTEGER       WGTNUM                          ! Max. weight number

      INTEGER       N4,N5,N6                        ! General Indices

      INTEGER       CLMNDX                          ! Huffman Column Index

      REAL*8        PRBTMP                          !   "       "     "

      INTEGER*4     CODTMP,LENTMP                   ! Temporary Coding Variables

      INTEGER*4     CODE(-1:34)                     ! Codeword Array

      INTEGER       CDWLEN(-1:34)                   ! Codeword Length Array

      REAL*8        PROB(-1:34)                     ! Probability Array

      INTEGER       NMBR(-1:34)                     ! Error Level Number Array

      INTEGER       POSN(-1:34)                     ! Column Position Array


      COMMON        /HUFBLK/PROB,CODE,WGTNUM,CDWLEN
```

```
**********************************************************************
*                                                                    *
* OBJECT: To perform a left-to-right reduction of the Huffman        *
*         coding table in order to locate the column positions       *
*         that determine the variable-length codewords.              *
*                                                                    *
* FUNC:           1. For QL-1 columns in Huffman table               *
*                 2.    Add last two entries in column               *
*                 3.    Determine its rank in next column            *
*                 4.    Place it in next column at that rank         *
*                 5.    Mark the position in POSN()                  *
*                 6. Go to next column                               *
*                                                                    *
**********************************************************************


      TBLNDX=WGTNUM
      DO N4=Ø,WGTNUM
          CLMNDX=Ø
          PRBTMP=PROB(TBLNDX)+PROB(TBLNDX+1)
          DO WHILE (PRBTMP.LT.PROB(CLMNDX))
              CLMNDX=CLMNDX+1
          END DO
          DO N5=TBLNDX,CLMNDX+1,-1
              PROB(N5)=PROB(N5-1)
          END DO
          PROB(CLMNDX)=PRBTMP
          POSN(N4)=CLMNDX
          TBLNDX=TBLNDX-1
      END DO
```

```
*********************************************************************
*                                                                   *
* OBJECT: To perform a right-to-left expansion of the Huffman       *
*         coding table in order to assign the variable-length       *
*         codewords based on the local conditional statistics       *
*         of the image.                                             *
*                                                                   *
* INPUT:  Column positions from the REDUCE routine.                 *
*                                                                   *
* OUTPUT: Variable-length codewords for the neighbor conditions.    *
*                                                                   *
* FUNC:        1. For QL-1 columns in Huffman table                 *
*              2.    Add a bit to codeword at next column's POSN()  *
*              3.    Set new bit of last entry in column to ZERO    *
*              4.    Set new bit of last entry in next column to ONE *
*              5. Go to next column                                 *
*                                                                   *
*********************************************************************


      TBLNDX=1
      CODE(Ø)=Ø
      CODE(1)=1
      CDWLEN(Ø)=1
      CDWLEN(1)=1
      DO N4=WGTNUM,Ø,-1
         DO N5=Ø,TBLNDX
            IF (POSN(N4).EQ.N5) THEN
               CODTMP=CODE(POSN(N4))
               LENTMP=CDWLEN(POSN(N4))
               DO N6=POSN(N4),TBLNDX-1
                  CODE(N6)=CODE(N6+1)
                  CDWLEN(N6)=CDWLEN(N6+1)
               END DO
               CODE(TBLNDX)=2*CODTMP
               CODE(TBLNDX+1)=2*CODTMP+1
               CDWLEN(TBLNDX)=LENTMP+1
               CDWLEN(TBLNDX+1)=CDWLEN(TBLNDX)
            END IF
         END DO
         TBLNDX=TBLNDX+1
      END DO
      RETURN
      END
```

Module HUFCOD   No errors   Prog: 323   Blank Common: None
FTN7X 244Ø/85Ø3Ø4   No warnings   Save: None   Local Ema: None

```
      INTEGER*4 FUNCTION BINOM(N,M)
      INTEGER   M,N
      REAL*6    T1,T2,T3,A
      INTEGER   I,B
      A = 1.0
      T1 = DBLE(N)
      IF ((2 * M).GT.N) THEN
         B = N - M
      ELSE
         B = M
      END IF
      DO I=0,B-1
         T2 = DBLE(I)
         A = (A * ((T1 - T2) / (T2 + 1.0)))
      END DO
      BINOM = DINT(A + 0.5)
      RETURN
      END
```

```
Module BINOM          No errors     Prog:   106    Blank Common: None
FTN7X  2440/850304    No warnings   Save:   None   Local Ema:    None
```

```
$FILES 2,2
      PROGRAM DECODE

C*********************************************************************
C     AUTHOR J. P. DIMAGGIO                                           *
C                                                                     *
C     DESCRIPTION:                                                    *
C                                                                     *
C     This program decodes the Encoded Conditioned Image File         *
C     comsisting of segment state words, back into the Conditioned    *
C     Image File.                                                     *
C                                                                     *
C                                                                     *
C                                                                     *
C   CALLING SEQUENCE:                                                 *
C                                                                     *
C      RU,DECODE,<INPUT NAMR>,<OUTPUT NAMR>                           *
C                                                                     *
Cooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
C
C    INPUT PARAMETERS:
C              NUMBER OF WORDS PER OUTPUT RECORD
C              NUMBER OF RECORDS PER OUTPUT FILE
C              NUMBER OF WORDS IN INPUT FILE
C              NUMBER OF RECORDS IN INPUT FILE
C              SEGMENT LENGTH TO BE COMPRESSED
C              MAXIMUM WEIGHT ENCODABLE FOR THAT SEGMENT
C
C*********************************************************************
C
      INCLUDE   DECODE.INC
C*********************************************************************
C
      IMPLICIT NONE
C
C*********************************************************************
C
C  DEFINE PARAMETERS
C
      INTEGER MAXLIN      ! MAXIMUM NO. OF OUTPUT RECORDS
      INTEGER DERECS      ! RECORDS IN INPUT FILE
      INTEGER BUFD        ! MAXIMUM NO. OF WORDS PER INPUT RECORD

      PARAMETER (MAXLIN=5000)
      PARAMETER (BUFD=225)
C
C************ FILE DEFINITIONS ***************************************
C
      INTEGER   ISTAT          ! I-O STATUS VARIABLE
      INTEGER   LBUF(BUFD)     ! READ/WRITE BUFFER FOR FORTRAN 77
      INTEGER   CSL(BUFD)      ! CSL BUFFER
      INTEGER   ESL(BUFD)      ! ENCODED SCAN LINE BUFFER

      INTEGER    LINMAX        ! NO. OF RECORDS TO BE OUTPUT
      INTEGER    BUFDIM        ! NO. OF WORDS PER OUTPUT RECORD
```

```
      INTEGER    MREC            ! NO. OF RECORDS IN INPUT FILE

      INTEGER   BUFMAX                 ! INPUT RECORD SIZE IN WORDS
C
C************* LOCAL VARIABLES ****************************************
C
      INTEGER*4 WEIGHT(-1:34)       ! HOLDS WEIGHTS
      INTEGER   CODLEN(-1:34)       ! HOLDS HUFF. CODE LENGTHS
      INTEGER   SEGBUF(BUFD)        ! SEGMENT BUFFER
      INTEGER   MAXWGT              ! MAX WEIGHT ENCODABLE OFR SEGMENT
      INTEGER   CTWRD               ! CURRENT WORD IN ESL
      INTEGER   CTCOD               ! CURRENT CODE IN CODE SEARCH
      INTEGER   CSLPOS              ! CURRENT POSITION IN CSL
      INTEGER   ESLPOS              ! CURRENT POSITION IN ESL
      INTEGER*4 SUB                 ! USED IN RANK DECODING PROCESS
      INTEGER   POS                 ! USED IN RANK DECODING PROCESS
      INTEGER   RANLEN              ! LENGTH OF RANK
      INTEGER*4 TOTAL               ! TOTAL NUMBER OF BITS COMPRESSED
      REAL      RATIO               ! COMPRESSION RATIO
      INTEGER   WRDS                ! NUMBER OF WORDS IN SEGMENT
      INTEGER   SEGS,SGCT           ! SEGMENTS IN A LINE AND SEGMENT COUNTER
      INTEGER   CHECK               ! WORD HOLDER FOR SEGBUF
      INTEGER   SEGWGT              ! CURRENT SEGMENT WEIGHT
      INTEGER*4 RANK                ! RANK OF SEGMENT
      INTEGER   LNONES              ! ONES SEEN IN LINE
      INTEGER*4 R1                  ! INTERMEDIATE RANK VALUE
      INTEGER   L1                  ! INTERMEDIATE RANK LENGTH
      INTEGER   NBPW                ! NUMBER OF BITS PER WORD
      REAL      LOG2                ! CONSTANT USED IN EQUATION
      INTEGER   NUMBER              ! NUMBER OF BITS IN A LINE
      INTEGER*4 BINOM               ! BINOMIAL FUNCTION RETURN
      INTEGER   LNWGT               ! HOLDS WEIGHT OF EACH CSL
      INTEGER   SEGL                ! SEGMENT LENGTH TO BE COMPRESSED
      INTEGER   SMLEN               ! SINGLE REP OF SEGL
      INTEGER*4 BITWRD              ! PRESENT WORD IN INPUT BUFFER
      INTEGER   OUTWRD              ! USED IN RANK DECODING
      INTEGER   CT                  ! INDEX OF CURRENT BIT IN LINE
      INTEGER   M,I,K,L             ! LOOP COUNTERS
C
C************* FILE PARAMETERS ****************************************
C
      CHARACTER INFILE*20
      CHARACTER OTFILE*20
      CHARACTER ACCTYP*20
      INTEGER   IRCL,JRCL,OREC  ! FILE PARAMETERS
      INTEGER   ITBUF(15)       ! ARRAY FOR TIME-OF-DAY
      INTEGER   TERM            ! TERMINAL LU
      INTEGER   LPFIL           ! PRINTER LU
      INTEGER   PELFIL          ! LU OF INPUT FILE
      INTEGER   OUTFIL          ! LU OF OUTPUT FILE
      INTEGER*4 I4B             ! LIBRARY ROUTINE
      INTEGER   ITLOG           ! LIBRARY FUNCTION

      LOGICAL   EXISTS
      LOGICAL   DNRANK          ! TRUE IF RANK DECODING IS FINISHED
```

```
      LOGICAL   DONE
      LOGICAL   FOUND
      EXTERNAL  BINOM             ! FUNCTION FOR COMPUTING BINOMIAL
                                    COEFFICIENT
      EXTERNAL  MB4B              ! DESCRIBED IN REPORT
      EXTERNAL  I4B               !  "          "

      COMMON   /ENCOD/WEIGHT,SEGL,NUMBER,CODLEN


      DATA TERM,LPFIL,PELFIL,OUTFIL/1,6,2,3/
```

```
C
C****************** BEGIN PROGRAM **********************************
C
      LOG2=LOG(2.Ø)

C
C   GET INPUT FILE NAME
C
      CALL RCPAR(1,INFILE)
      INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
     &        EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
      IF(ISTAT.NE.Ø) STOP 'INQUIRE ERROR'
      IF(EXISTS) THEN
         IF(ACCTYP.EQ.'SEQUENTIAL') THEN
            IF(IRCL.EQ.Ø.AND.MREC.EQ.Ø) THEN
               OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
               CALL LGBUF(LBUF,BUFD)
               READ(UNIT=PELFIL,IOSTAT=ISTAT) (CSL(I),I=1,BUFD+1)
               IF(ISTAT.NE.Ø) THEN
                  IRCL=ITLOG()
                  MREC=MAXLIN
                  BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
                  IF(ISTAT.NE.Ø) STOP 'BACKSPACE ERROR'
               ELSE
                  STOP 'NO ERROR IS AN ERROR'
               END IF
            ELSE
               STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
            END IF
         ELSE IF(ACCTYP.EQ.'DIRECT') THEN
            OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT'
     &           ,RECL=IRCL,MAXREC=MREC)
         ELSE
            STOP 'ACCTYP UNDEFINED'
         END IF
      ELSE
         STOP 'INPUT FILE DOES NOT EXIST'
      END IF
C
C********** READ IN INPUT PARAMETERS ********************************
C
      BUFDIM=Ø
      DO WHILE (BUFDIM.LT.1.OR.BUFDIM.GT.BUFD)
        WRITE(TERM,'(''$ENTER NUMBER OF WORDS PER OUTPUT RECORD: '')')
        READ(TERM,*) BUFDIM
      END DO

      LINMAX=Ø
      DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
        WRITE(TERM,'(''$NUMBER OF RECORDS TO BE OUTPUT = :'')')
        READ(TERM,*) LINMAX
      END DO
C
C********** GET OUTPUT FILE NAME AND OPEN ****************************
C
```

```
      CALL RCPAR(2,OTFILE)
      OREC=MIN0(MREC,LINMAX)
      JRCL=BUFDIM*2
      OPEN(UNIT=OUTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRCL
     &    ,MAXREC=OREC,STATUS='NEW',IOSTAT=ISTAT)
      IF(ISTAT.NE.0) THEN
         WRITE(TERM,*) 'OUTPUT FILE OPEN ERROR',ISTAT
      END IF

      WRITE(TERM,*) 'NUMBER OF BITS PER WORD?'
      READ(TERM,*) NBPW
C
C************* WRITE INPUT PARAMETERS ********************************
C
      CALL FTIME(ITBUF)
      WRITE(LPFIL,'(1H0,15A2)') ITBUF
      WRITE(LPFIL,'(''0INPUT FILE NAME - '',A20)') INFILE
      WRITE(LPFIL,'(''0OUTPUT FILE NAME - '',A20)') OTFILE

      WRITE(LPFIL,200) BUFDIM,OREC,IRCL,MREC,JRCL,NBPW
  200 FORMAT('0INPUT PARAMETERS:'/
     *          ' NUMBER OF WORDS PER OUTPUT RECORD =',I6/
     *          ' NUMBER OF RECORDS TO BE WRITTEN =',I6/
     *          ' INPUT RECORD SIZE ='I7' BYTES'/
     *          ' NO. OF INPUT RECORDS ='I7/
     *          ' OUTPUT RECORD SIZE ='I7' BYTES'/
     *          ' NUMBER OF BITS PER WORD='I7)


       WRITE(TERM,*) 'WORDS PER INPUT RECORD?'
       READ(TERM,*) BUFMAX
       NUMBER=BUFMAX*16
       WRITE(TERM,*) 'NUMBER OF RECORDS IN INPUT FILE?'
       READ(TERM,*) DERECS
       MREC=MIN(MREC,DERECS)
       WRITE(TERM,*) 'SEGMENT LENGTH TO BE COMPRESSED?'
       READ(TERM,*) SEGL
       WRITE(TERM,*) 'MAX WEIGHT ENCODABLE?'
       READ(TERM,*) MAXWGT
C
C*********** GET RECORDS TO PROCESS ***********************************
C
      WRDS=SEGL/16+1
      WRITE(LPFIL,*) 'SEGMENT LENGTH COMPRESSED= ',SEGL
      WRITE(LPFIL,*) 'MAX WEIGHT= ',MAXWGT
      SMLEN=SEGL
      SEGS=NUMBER/SMLEN+1
      TOTAL=0
      DO I=1,MREC
        CALL LGBUF(LBUF,BUFD)
        READ(UNIT=PELFIL,IOSTAT=ISTAT)(ESL(M),M=1,BUFMAX)
        DO L=1,BUFMAX
           CSL(L)=0
        END DO
        IF (.NOT.(BTEST(ESL(1),15))) THEN
```

```
      LNWGT=IBITS(ESL(1),3,13)
      CALL CODGEN(LNWGT,MAXWGT)
      DONE=.FALSE.
      CSLPOS=1
      SGCT=0
      LNONES=0
      CTWRD=1
      ESLPOS=13
      DO WHILE(.NOT.DONE)
         FOUND=.FALSE.
         SGCT=SGCT+1
C
C************ SEARCH FOR HUFFMAN CODE ********************************
C
         BITWRD=I48(ESL,ESLPOS+1,32)
         CTCOD=0
         DO WHILE(.NOT.FOUND)
            IF ((IBITS(BITWRD,32-CODLEN(CTCOD),CODLEN(CTCOD)))
     &          .EQ.(IBITS(WEIGHT(CTCOD),0,CODLEN(CTCOD)))) THEN

               FOUND=.TRUE.
               SEGWGT=CTCOD
            ELSE
               CTCOD=CTCOD+1
            END IF
         END DO
         ESLPOS=ESLPOS+CODLEN(CTCOD)
         IF((SEGWGT.EQ.0).OR.(SEGWGT.EQ.MAXWGT+1)) THEN
            IF (SEGWGT.EQ.0) THEN
               DO L=1,WRDS
                  SEGBUF(L)=0
               END DO
            ELSE
               CALL MB4B(SEGBUF,1,SMLEN,ESL,ESLPOS+1)
               ESLPOS=ESLPOS+SMLEN
               M=0
               CT=0
               DO WHILE(CT.LT.SMLEN)
                  IF(MOD(CT,16).EQ.0) THEN
                     M=M+1
                     CHECK=SEGBUF(M)
                     K=16
                  END IF
                  K=K-1
                  CT=CT+1
                  IF(BTEST(CHECK,K)) THEN
                     LNONES=LNONES+1
                  END IF
               END DO
            END IF
         ELSE
            LNONES=LNONES+SEGWGT
            RANLEN=NINT((LOG(REAL(BINOM(SEGL,SEGWGT)))
     &             /LOG2)+0.5)
            IF ((NBPW.EQ.32).AND.(RANLEN.GT.16)) THEN
```

```
            R1=0
            L1=MOD(RANLEN,16)
            R1=I4B(ESL,ESLPOS+1,L1)
            CALL MVBITS(R1,0,L1,RANK,16)
            RANLEN=RANLEN-L1
            ESLPOS=ESLPOS+L1
         END IF
         R1=I4B(ESL,ESLPOS+1,RANLEN)
         CALL MVBITS(R1,0,RANLEN,RANK,0)
         ESLPOS=ESLPOS+RANLEN
         CT=16
         POS=SEGL
         OUTWRD=0
         DNRANK=.FALSE.
         M=0
         DO WHILE((POS.GT.0).AND.(.NOT.DNRANK))
            POS=POS-1
            CT=CT-1
            IF(SEGWGT.EQ.(POS+1)) THEN
               DNRANK=.TRUE.
               DO L=POS,0,-1
                  IF(CT.LT.0) THEN
                     M=M+1
                     SEGBUF(M)=OUTWRD
                     OUTWRD=0
                     CT=15
                  END IF
                  OUTWRD=IBSET(OUTWRD,CT)
                  CT=CT-1
               END DO
            ELSE
               SUB=BINOM(POS,SEGWGT)
               IF ((RANK-SUB).GE.0) THEN
                  RANK=RANK-SUB
                  SEGWGT=SEGWGT-1
                  OUTWRD=IBSET(OUTWRD,CT)
               END IF
            END IF
            IF(CT.LE.0) THEN
               M=M+1
               SEGBUF(M)=OUTWRD
               OUTWRD=0
               CT=16
            END IF
         END DO
         M=M+1
         SEGBUF(M)=OUTWRD
      END IF
      CALL MB4B(CSL,CSLPOS,SMLEN,SEGBUF,1)
      CSLPOS=CSLPOS+SMLEN
      IF((LNONES.GE.LNWGT).OR.(SEGS.EQ.SGCT)) THEN
         DONE=.TRUE.
      END IF
   END DO
   TOTAL=TOTAL+ESLPOS
```

```
            ELSE
               TOTAL=TOTAL+1
            END IF
            CALL LGBUF(LBUF,BUFD)
            WRITE(UNIT=OUTFIL,IOSTAT=ISTAT) (CSL(M),M=1,BUFMAX)
         END DO
         RATIO=REAL(NUMBER)*MREC/REAL(TOTAL)
         WRITE(LPFIL,*) 'COMPRESSION RATIO= ',RATIO
     250 FORMAT(10(I10))

         CLOSE(UNIT=PELFIL)
         CLOSE(UNIT=OUTFIL)

C*****************************************************************
C                                                                *
C   END OF PROGRAM                                               *
C                                                                *
C*****************************************************************

         END
```

```
Module DECODE        No errors     Prog:  2881    Blank Common: None
FTN7X  2440/850304   No warnings   Save:  None    Local Ema:    None
```

```
$FILES 3,4,5
      PROGRAM RESTORE

C**********************************************************************
C
C     AUTHOR J. P. DIMAGGIO
C
C     PROGRAM LAST CHANGED   <850814.1140>
C
C     DESCRIPTION:
C
C     This program restores the Conditioned Image File
C     back to the original file after decoding.
C
C
C   CALLING SEQUENCE:
C
C      RU,RESTORE,<INPUT NAMR>,<OUTPUT NAMR>,<PROBABILITY FILE>
C
Cooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
C
C    INPUT PARAMETERS:
C              NUMBER OF WORDS PER OUTPUT RECORD
C              NUMBER OF RECORDS PER OUTPUT FILE
C              NUMBER OF RECORDS IN INPUT FILE
C
C************************************************************************
C
      IMPLICIT NONE
C
C************************************************************************
C
C  DEFINE PARAMETERS
C
      INTEGER MAXLIN       ! MAXIMUM NO. OF OUTPUT RECORDS
      INTEGER DERECS       ! RECORDS IN INPUT FILE
      INTEGER BUFD         ! MAXIMUM NO. OF WORDS PER INPUT RECORD
      INTEGER BITS         ! MAXIMUM NO. OF BITS PER RECORD RECORD

      PARAMETER (MAXLIN=5000)
      PARAMETER (BITS=4100)
      PARAMETER (BUFD=225)
C
C************ FILE DEFINITIONS *****************************************
C
      INTEGER   ISTAT           ! I-O STATUS VARIABLE
      INTEGER   LBUF(BUFD)      ! READ/WRITE BUFFER FOR FORTRAN 77
      INTEGER   BUFFER(BUFD)    ! RECORD BUFFER
      INTEGER   OBUFF(BUFD)     ! OUTPUT BUFFER

      INTEGER    LINMAX  ! NO. OF RECORDS TO BE OUTPUT
      INTEGER    BUFDIM  ! NO. OF WORDS PER OUTPUT RECORD
      INTEGER    MREC    ! NO. OF RECORDS IN INPUT FILE

      INTEGER   BUFMAX                 ! INPUT RECORD SIZE IN WORDS
```

```
C
C************* LOCAL VARIABLES ****************************************
C
      INTEGER   PREV(-1:BITS)        ! HOLDS BIT VALUES OF PREVIOUS LINE
      INTEGER   PROB(16)             ! DATA TO BE USED IN CALCULATIONS
      INTEGER   NBR(4)               ! HOLDS VALUES OF NEIGHBORING BITS
      INTEGER   BITVAL               ! HOLDS VALUE OF CURRENT BIT
      INTEGER   PREBIT               ! HOLDS VALUE OF PREVIOUS BIT
      INTEGER   PSHOLD               ! HOLDS POSITION OF 1 BIT IN CSL
      INTEGER   BITWRD               ! PRESENT WORD IN INPUT BUFFER
      INTEGER   OBTWRD               ! PRESENT WORD IN OUTPUT BUFFER
      INTEGER   CT                   ! INDEX OF CURRENT BIT IN LINE
      INTEGER   P,I,K,L,Q            ! LOOP COUNTERS
      INTEGER   FIND,MUL             ! NARROWING VARIABLES IN SEARCH
C
C************* FILE PARAMETERS ****************************************
C
      CHARACTER INFILE*20
      CHARACTER OTFILE*20
      CHARACTER DATFIL*20
      CHARACTER ACCTYP*20
      INTEGER   IRCL,JRCL,OREC   ! FILE PARAMETERS
      INTEGER   ITBUF(15)        ! ARRAY FOR TIME-OF-DAY
      INTEGER   TERM             ! TERMINAL LU
      INTEGER   LPFIL            ! PRINTER LU
      INTEGER   PELFIL           ! LU OF INPUT FILE
      INTEGER   OUTFIL           ! LU OF OUTPUT FILE
      INTEGER   DTFIL            ! LU OF DATA FILE
      INTEGER   ITLOG            ! LIBRARY FUNCTION

      LOGICAL   EXISTS

      DATA TERM,LPFIL,PELFIL,OUTFIL,DTFIL/1,6,2,3,4/
```

```
C
C************ INITIALIZING ******************************************
C
      DO I=-1,BITS
         PREV(I)=0
      END DO
C
C****************** BEGIN PROGRAM ***********************************
C
      CALL RCPAR(1,INFILE)
      INQUIRE(FILE=INFILE,RECL=IRCL,MAXREC=MREC,
     &        EXIST=EXISTS, ACCESS=ACCTYP,IOSTAT=ISTAT)
      IF(ISTAT.NE.0) STOP 'INQUIRE ERROR'
      IF(EXISTS) THEN
         IF(ACCTYP.EQ.'SEQUENTIAL') THEN
            IF(IRCL.EQ.0.AND.MREC.EQ.0) THEN
               OPEN(UNIT=PELFIL,FILE=INFILE,ACCESS=ACCTYP)
               CALL LGBUF(LBUF,BUFD)
               READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(I),I=1,BUFD+1)
               IF(ISTAT.NE.0) THEN
                  IRCL=ITLOG()
                  MREC=MAXLIN
                  BACKSPACE(UNIT=PELFIL,IOSTAT=ISTAT)
                  IF(ISTAT.NE.0) STOP 'BACKSPACE ERROR'
               ELSE
                  STOP 'NO ERROR IS AN ERROR'
               END IF
            ELSE
               STOP 'NOT A DIR. ACC. FILE NOR MAGTAPE'
            END IF
         ELSE IF(ACCTYP.EQ.'DIRECT') THEN
            OPEN(UNIT=PELFIL,FILE=INFILE,STATUS='OLD',ACCESS='DIRECT'
     &           ,RECL=IRCL,MAXREC=MREC)
         ELSE
            STOP 'ACCTYP UNDEFINED'
         END IF
      ELSE
         STOP 'INPUT FILE DOES NOT EXIST'
      END IF
C
C**************** READ IN INPUT PARAMETERS **************************
C
      BUFDIM=0
      DO WHILE (BUFDIM.LT.1.OR.BUFDIM.GT.BUFD)
        WRITE(TERM,'(''$ENTER NUMBER OF WORDS PER OUTPUT RECORD: '')')
        READ(TERM,*) BUFDIM
      END DO

      LINMAX=0
      DO WHILE (LINMAX.LT.1.OR.LINMAX.GT.MAXLIN)
        WRITE(TERM,'(''$NUMBER OF RECORDS TO BE OUTPUT = :'')')
        READ(TERM,*) LINMAX
      END DO
C
C**************** GET OUTPUT FILE NAME AND OPEN *********************
```

```
C
      CALL RCPAR(2,OTFILE)
      OREC=MIN0(MREC,LINMAX)
      JRCL=BUFDIM*2
      OPEN(UNIT=OUTFIL,FILE=OTFILE,ACCESS='DIRECT',RECL=JRCL
     &     ,MAXREC=OREC,STATUS='NEW',IOSTAT=ISTAT)
      IF(ISTAT.NE.0) STOP 'OUTPUT FILE OPEN ERROR'
C
C************** WRITE INPUT PARAMETERS *******************************
C
      CALL FTIME(ITBUF)
      WRITE(LPFIL,'(1H0,15A2)') ITBUF
      WRITE(LPFIL,'(''0INPUT FILE NAME - '',A20)') INFILE

      WRITE(LPFIL,200) BUFDIM,OREC,IRCL,MREC,JRCL
  200 FORMAT('0INPUT PARAMETERS:'/
     *        ' NUMBER OF WORDS PER OUTPUT RECORD =',I6/
     *        ' NUMBER OF RECORDS TO BE WRITTEN =',I6/
     *        ' INPUT RECORD SIZE ='I7' BYTES'/
     *        ' NO. OF INPUT RECORDS ='I7/
     *        ' OUTPUT RECORD SIZE ='I7' BYTES')
C
C*********** READ IN PROBABILITY DATA ********************************
C
       CALL RCPAR(3,DATFIL)
       OPEN(UNIT=DTFIL,FILE=DATFIL)
       DO I=1,16
         READ(DTFIL,*) PROB(I)
       END DO
C
C*********** GET RECORDS TO PROCESS **********************************
C
      WRITE(TERM,*)('NUMBER OF RECORDS IN INPUT FILE::')
      READ(TERM,*) DERECS
      MREC=MIN(MREC,DERECS)
C
C*********** BEGIN FILE PROCESSING ***********************************
C
      BUFMAX=IRCL/2

      DO I=1,MREC
           CT=1
           PSHOLD=0
           PREBIT=0

           CALL LGBUF(LBUF,BUFD)
           READ(UNIT=PELFIL,IOSTAT=ISTAT) (BUFFER(P),P=1,BUFMAX)
           IF(ISTAT.NE.0) THEN
             WRITE(TERM,'(''INPUT READ ERROR, ISTAT='',I4)') ISTAT
             STOP
           END IF

           DO K=1,BUFMAX
                 BITWRD=BUFFER(K)
                 OBTWRD=0
```

```

                DO L=15,0,-1
                   IF (BTEST(BITWRD,L))THEN
                    BITVAL=1
                   ELSE
                       BITVAL=0
                   END IF
                   DO Q=1,3
                      NBR(Q)=PREV((Q-1)+(CT-1))
                   END DO
                   NBR(4)=PREBIT
                   PREV(CT-1)=PREBIT
                   FIND=1
                   MUL=16
                   DO P=1,4
                      IF (NBR(P).EQ.1) THEN
                         FIND=FIND+MUL/2
                      END IF
                      MUL=MUL/2
                   END DO

                   IF (BITVAL.EQ.0) THEN
                      IF (PROB(FIND).EQ.1) THEN
                         OBTWRD=IBSET(OBTWRD,L)
                         PREBIT=1
                      ELSE
                         PREBIT=0
                      END IF
                   ELSE
                      IF (PROB(FIND).EQ.0) THEN
                        OBTWRD=IBSET(OBTWRD,L)
                        PREBIT=1
                      ELSE
                        PREBIT=0
                      END IF
                   END IF
                   CT=CT+1
                END DO      ! NO BITS IN WORD

            OBUFF(K)=OBTWRD
            END DO     ! NO WORDS IN RECORD
            PREV(CT-1)=PREBIT
C
C************* WRITE RECORD TO OUTPUT FILE ***************************
C
        CALL LGBUF(LBUF,BUFD)
        WRITE(UNIT=OUTFIL,IOSTAT=ISTA1) (OBUFF(Q),Q=1,BUFDIM)
        IF( ISTAT.NE.0) THEN
          WRITE(TERM,'(''OUTPUT READ ERROR, ISTAT='',I4)') ISTAT
          STOP
        END IF
        END DO ! RECORDS IN FILE

C*******************************************************************
C                                                                   *
```

```
C         END OF PROGRAM                                                 *
C                                                                        *
C*************************************************************************

      END
```

```
Module RESTORE            No errors     Prog:  8619    Blank Common: None
FTN7X  2440/850304        No warnings   Save:  None    Local Ema:    None
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704 0188
Exp Date Jun 30 1986

AD-A160846

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT: Unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S): NCS TIB 85-6
6a NAME OF PERFORMING ORGANIZATION: Delta Information Systems, Inc. — 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION: National Communications System
6c ADDRESS (City, State, and ZIP Code): Horsham Business Center, Bldg 3, 300 Welsh Road, Horsham, PA 19044	7b ADDRESS (City, State, and ZIP Code): ATTN: NCS-TS, Washington, DC 20305-2010
8a NAME OF FUNDING / SPONSORING ORGANIZATION: National Communications System — 8b OFFICE SYMBOL (If applicable): NCS-TS	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: Contract DCA100-83-C-0047
8c ADDRESS (City, State and ZIP Code): ATTN: NCS-TS, Washington, DC 20305-2010	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 33127K; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

Scan Line Difference Compression Algorithm Simulation Study

12 PERSONAL AUTHOR(S)

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM ____ TO ____	85-08-23	84

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Facsimile, Modread, Compression Algorithm

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this task was to investigate the efficiency and potential operational effectiveness of the Scan Line Difference Compression (SLDC) algorithm presented in Appendix A of the report. Delta Information Systems (DIS) performed a software simulation study of the SLDC algorithm. The software was run using selected CCITT binary standard images as input, and the results were compared with those of the MODREAD II compression algorithm run with the same input data. The MODREAD II algorithm was chosen for the comparison because it is the most effective compression technique currently available.

Keywords: ...

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) — 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

END

FILMED

12-85

DTIC